Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Aplicaciones con Tecnología en Internet

Desarrollo de un Módulo de generación de reportes para el sistema CONEST de la Facultad de Ciencias

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela
por los Bachilleres
María Eugenia Márquez Colletti y **Wilmer Antonio Fernández Pestana**
para optar al título de Licenciado en Computación

Tutor
Prof. Sergio Rivas
Caracas, Junio / 2009.

ACTA

Quienes suscriben miembros del Jurado, designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por los Bachilleres María Eugenia Máquez Colletti, C.I. 17.146.056, y Wilmer Antonio Fernández Pestana, C.I. 17.388.219, con el título “Desarrollo de un módulo de generación de reportes para el sistema CONEST de la Facultad de Ciencias”, a los fines de optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijó el 08 de junio de 2009 a las 11:00 a.m., para que sus autores lo defendieran en forma pública. Se hizo en la sala de Postgrado de la Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, mediante una presentación oral de su contenido. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlo con una nota de puntos, en fe de lo cual se levanta la presente Acta, en Caracas a los ochos días del mes de junio del año dos mil nueve, dejándose también constancia de que actuó como Coordinador del Jurado el profesor Sergio Rivas.

_____

Prof. Sergio Rivas
Tutor

_____ _____

Prof. Adrian Bottini
Jurado Principal

Prof. Andrés Sanoja
Jurado Principal

_____ _____

Profa. Jossie Zambrano
Jurado Suplente

Profa. Damaris Barrantes
Jurado Suplente

AGRADECIMIENTOS

Gracias a Dios, por ser mi guía y mi más fiel amigo todos estos años, por permitirme estar en donde me encuentro ahora, por darme luz en los momentos oscuros y fuerzas cuando perdía el ánimo.

Gracias a mis padres, por todo lo que me han dado, por su cariño incondicional, por apoyarme, por ayudarme en todo lo que pudieron y por tratar de entender todas estas cosas que quizás aún no comprendan.

Gracias a mi nonna y a mis hermanos que de alguna u otra forma, directa o indirectamente también estuvieron allí para ayudarme.

Gracias Wilmer por estar siempre a mi lado, en las buenas y en las malas, por darme tu apoyo y ser siempre constante.

Gracias a esta hermosa casa de estudios, en la que tuve la oportunidad de formarme como profesional y a todos los profesores que participaron en este proceso.

María Eugenia.

AGRADECIMIENTOS

Gracias a Dios, por ayudarme a enfrentar todos esos momentos difíciles y darme fuerzas para no rendirme, superar cada obstáculo y lograr mis metas.

Gracias a mi familia, por darme todo lo que he necesitado en esta vida, gracias a ella nunca me ha faltado absolutamente nada, siempre he tenido su total apoyo en todos los aspectos, bien sea en el sentimental, económico, etc., para poder llegar hasta este logro, que sin duda alguna no hubiese podido ser realidad sin ustedes.

Gracias María Eugenia, por estar siempre conmigo apoyándome y darme la oportunidad de compartir juntos durante toda la carrera tantas aventuras, experiencias, desveladas, éxitos y frustraciones.

Gracias a la UCV y cada uno de los profesores que participaron en mi desarrollo profesional durante mi carrera, sin su ayuda y conocimientos no estaría en donde me encuentro ahora.

Wilmer.

RESUMEN

Actualmente la División de Control de Estudios de la Facultad de Ciencias de la Universidad Central de Venezuela dispone de una aplicación Web denominada CONEST, la cual se encarga de automatizar sus procesos y a su vez almacena toda la información correspondiente a la gestión académica de la Facultad. Para los distintos procesos que lleva a cabo la División de Control de Estudios es necesario emitir una serie de reportes, para los cuales no poseen una herramienta que les permita elaborarlos, además actualmente el modelo de datos de este sistema posee alrededor de 140 tablas y esto dificulta la tarea de elaboración de reportes.

Dada esta situación, en el presente Trabajo Especial de Grado se desarrolló una aplicación, la cual es a su vez un módulo del sistema CONEST, que está dedicado exclusivamente a la generación y administración de reportes a partir de los datos almacenados en el repositorio de datos, todo esto basándose en el estudio de algunos generadores de reportes comerciales, analizando la forma como estos, a través de secuencias ordenadas de pasos (wizards), apoyan al usuario en la tarea de elaboración de reportes. Entre los aspectos de desarrollo de esta aplicación tenemos el diseño del wizard que permite la creación de los reportes de una forma más sencilla, el desarrollo de los algoritmos de exportación de datos a los formatos soportados, el análisis del modelo de datos actual de CONEST, en base al cual se diseñaron un conjunto de vistas que faciliten el acceso a los datos y la integración con el sistema CONEST.

Palabras claves

Reporte, Ruby on Rails, Wizard, MySQL, Vistas.

Contactos

- María Eugenia Márquez: mariumarquez@gmail.com
- Wilmer Fernández: wilmerfernandez@gmail.com
- Sergio Rivas: srivas@gmail.com
- Jossie Zambrano: jossie.zambrano@gmail.com

ÍNDICE GENERAL

INTRODUCCIÓN 17
PROPUESTA DE TRABAJO ESPECIAL DE GRADO 19
1 PROPUESTA DE TRABAJO ESPECIAL DE GRADO 21
1.1 Descripción del Problema 21
1.2 Objetivo General 25
1.3 Objetivos Específicos 26
1.4 Proceso de Desarrollo 26
1.5 Tecnologías y Plataformas 26
MARCO CONCEPTUAL 29
2 HERRAMIENTAS PARA ELABORAR REPORTES 31
2.1 Los reportes y las herramientas para elaborarlos 31
2.2 Crystal Reports XI Professional 33
2.2.1 Características 33
2.2.2 Proceso de generación de reportes 33
2.2.3 Evaluación de la Herramienta 40
2.3 Navicat 8 43
2.3.1 Características 43
2.3.2 Proceso de generación de reportes 43
2.3.3 Evaluación de la Herramienta 51
2.4 SQL Manager for MySQL 2007 53
2.4.1 Características 53
2.4.2 Proceso de generación de reportes 53
2.4.3 Evaluación de la Herramienta 57
2.5 Cuadro comparativo de las herramientas 59
2.6 El patrón de diseño: Wizard 61
2.6.1 Los wizards y su uso 61
2.6.2 Consideraciones de diseño de los wizards 62
3 IDENTIFICACIÓN DE LOS REPORTES 65
3.1 Análisis de la situación actual 65
3.2 Entidades 67
3.2.1 Entidad Estudiante 68
3.2.2 Entidad Materia 69
3.2.3 Entidad Docente 69
3.3 Clasificaciones de los Reportes 71
3.3.1 Reportes Regulares 71
3.3.2 Reportes por Solicitud 71
3.3.3 Listados 71
3.3.4 Tabla Cruzada 72
3.4 El Sistema Manejador de Base de Datos MySQL 77
3.4.1 El manejo de Vistas 78
MARCO APLICATIVO 83
4 ADAPTACIÓN XP 85
4.1 Adaptación del Proceso de Desarrollo XP 85
4.1.1 Programación Extrema 85
4.1.2 Iteración Programación Extrema 85
4.1.3 Historias de Usuarios 86
4.1.4 Adaptación de las Tareas XP 87
4.1.5 Actores y Responsabilidades 88
4.2 Metáfora del Sistema 89
4.3 Requerimientos Generales del Sistema 91
4.3.1 Requerimientos Funcionales 91
4.3.2 Requerimientos No Funcionales 91
5 DESARROLLO DE LA APLICACIÓN 95

5.1 Iteración 0 95
5.2 Iteración 1 99
5.3 Iteración 2 103
5.4 Iteración 3 109
5.5 Iteración 4 113
5.6 Iteración 5 121
5.7 Iteración 6 127
5.8 Iteración 7 135
5.9 Iteración 8 143
5.10 Iteración 9 151
5.11 Iteración 10 159
5.12 Iteración 11 163
5.13 Iteración 12 169
5.14 Iteración 13 175
CONCLUSIONES 181
RECOMENDACIONES 185
REFERENCIAS 187
APÉNDICE 189
A ENCUESTA DE USABILIDAD 191
B CASO DE ESTUDIO 195
B.1 Ambiente y especificación de las pruebas 195
B.2 Prueba 1 197
B.3 Prueba 2 203
B.4 Prueba 3 209
B.5 Prueba 4 213
B.6 Prueba 5 219
B.7 Prueba 6 223

ÍNDICE DE FIGURAS

Figura Nº 1-1. Propuesta. 24
Figura Nº 1-2. Partes o zonas de un Reporte. 25
Figura Nº 2-1. Resultado de un reporte de tabla cruzada en Crystal Reports. 34
Figura Nº 2-2. Selección de campos en Crystal Reports. 35
Figura Nº 2-3. Selección de diseño en Crystal Reports. 36
Figura Nº 2-4. Resultado de un reporte estándar en Crystal Reports. 37
Figura Nº 2-5. Relaciones entre las tablas en Crystal Reports. 38
Figura Nº 2-6. Disposición de los campos en la tabla del reporte de Tabla Cruzada de Crystal Reports. 39
Figura Nº 2-7. Query Designer de Navicat. 44
Figura Nº 2-8. Selección de tablas para el reporte de listado en Navicat. 45
Figura Nº 2-9. Selección manual de los campos que deben relacionarse en el reporte de listado en Navicat. 45
Figura Nº 2-10. Tablas y campos seleccionados en el reporte de listado en Navicat. 46
Figura Nº 2-11. Definición de restricciones en el reporte en Navicat. 46
Figura Nº 2-12. Vista preliminar de los resultados en Navicat. 47
Figura Nº 2-13. Opciones para generar un reporte en Navicat. 47
Figura Nº 2-14. Selección de campos en el reporte de listado en Navicat. 48
Figura Nº 2-15. Seleccionando un estilo en el reporte de listado en Navicat. 48
Figura Nº 2-16. Resultado del reporte de listado en Navicat. 49
Figura Nº 2-17. Disposición de los campos en la tabla del reporte de tabla cruzada de Navicat. 50
Figura Nº 2-18. Reporte de tabla cruzada exportado a formato PDF. 51
Figura Nº 2-19. Definición de la consulta en SQL Manager for MySQL 2007. 54
Figura Nº 2-20. Selección de campos en SQL Manager for MySQL 2007. 55
Figura Nº 2-21. Resultado del reporte de listado en SQL Manager for MySQL 2007. 55
Figura Nº 2-22. Ejemplo de uso del patrón en el sistema operativo Windows. 61
Figura Nº 2-23. Uso del patrón Wizard en la interfaz de Crystal Reports. 63
Figura Nº 3-1. Proceso actual de generación de reportes. 66
Figura Nº 3-2. Entidad Estudiante. 68
Figura Nº 3-3. Entidad Materia. 69
Figura Nº 3-4. Entidad Docente. 69
Figura Nº 3-5. Ejemplo de Reporte de Listado. 72
Figura Nº 3-6. Ejemplo de Reporte de Tabla Cruzada. 72
Figura Nº 3-7. Resultado de consultar la vista creada en el ejemplo. 79
Figura Nº 4-1. Desarrollo en XP. 87
Figura Nº 4-2. Metáfora del sistema. 89
Figura Nº 5-1. Diagrama entidad-relación del sistema. 96
Figura Nº 5-2. Estructura del campo xml_reporte. 97
Figura Nº 5-3. Diagrama de clases del sistema. 100
Figura Nº 5-4. Listado de reportes. 100
Figura Nº 5-5. Código de búsqueda de un reporte. 101
Figura Nº 5-6. Código de la opción Ver mis reportes. 102
Figura Nº 5-7. Código de la opción Ver reportes de mi rol. 102
Figura Nº 5-8. Diagrama de la clase Controlador. 104
Figura Nº 5-9. Diagrama de la clase Reporte. 105
Figura Nº 5-10. Código del método crear_reporte_sesion. 106
Figura Nº 5-11. Código del método validar_consulta. 107
Figura Nº 5-12. Código del método guardar_reporte. 108
Figura Nº 5-13. Diagrama de la clase Exportador. 109
Figura Nº 5-14. Diagrama de la clase Controlador. 110
Figura Nº 5-15. Diagrama de las vistas creadas en el repositorio de datos. 110
Figura Nº 5-16. Código del método exportar_csv. 111
Figura Nº 5-17. Diagrama de la clase Controlador. 114

Figura Nº 5-18. Diagrama de la clase Reporte. 114
Figura Nº 5-19. Vista paso 3. 115
Figura Nº 5-20. Vista paso 4. 116
Figura Nº 5-21. Código del método entidades_con_atributos. 117
Figura Nº 5-22. Código del método paso_4. 118
Figura Nº 5-23. Código del método cargar_campos_XML. 119
Figura Nº 5-24. Diagrama de las vistas incorporadas al repositorio de datos. 122
Figura Nº 5-25. Diagrama de la clase Controlador. 123
Figura Nº 5-26. Diagrama de la clase Reporte. 123
Figura Nº 5-27. Vista paso 5. 124
Figura Nº 5-28. Código del método paso_6. 125
Figura Nº 5-29. Código del método cargar_ordenamiento_Xml. 126
Figura Nº 5-30. Diagrama de la clase Exportador. 127
Figura Nº 5-31. Código del primer fragmento del método exportar_excel. 128
Figura Nº 5-32. Código del segundo fragmento del método exportar_excel. 129
Figura Nº 5-33. Código del primer fragmento del método exportar_pdf. 130
Figura Nº 5-34. Código del método max_columns. 131
Figura Nº 5-35. Código del segundo fragmento del método exportar_pdf. 132
Figura Nº 5-36. Código del tercer fragmento del método exportar_pdf. 132
Figura Nº 5-37. Código del cuarto fragmento del método exportar_pdf. 133
Figura Nº 5-38. Diagrama de la clase Reporte. 136
Figura Nº 5-39. Código del método agregar_campo. 137
Figura Nº 5-40. Código del método quitar_campo. 138
Figura Nº 5-41. Código del método mover_arriba. 139
Figura Nº 5-42. Código del método mover_abajo. 140
Figura Nº 5-43. Código del método agregar_restriccion. 141
Figura Nº 5-44. Diagrama de la clase Reporte. 144
Figura Nº 5-45. Vista paso 6. 146
Figura Nº 5-46. Código del método agregar_orden. 147
Figura Nº 5-47.Código del método verificar_orden_operaciones_arreglos. 148
Figura Nº 5-48.Código del método agregar_funcion. 148
Figura Nº 5-49.Código de un fragmento del método consulta_conteo. 149
Figura Nº 5-50. Diagrama de la clase Reporte. 152
Figura Nº 5-51. Diagrama de la clase Exportador. 153
Figura Nº 5-52. Vista paso 3 Tabla Cruzada. 154
Figura Nº 5-53. Vista paso 5 Definición de orden y funciones. 155
Figura Nº 5-54. Código del método agregar_funcion_conteo. 156
Figura Nº 5-55. Código del método imprimir_conteo_csv. 157
Figura Nº 5-56. Diagrama con nueva vista incorporada al repositorio de datos. 160
Figura Nº 5-57. Nueva vista paso 3. 161
Figura Nº 5-58. Nueva vista paso 4. 161
Figura Nº 5-59. Nueva vista paso 3 Tabla Cruzada. 162
Figura Nº 5-60. Nueva mejora a la vista paso 3. 164
Figura Nº 5-61. Nueva mejora a la vista paso 4. 164
Figura Nº 5-62. Nueva mejora a la vista paso 3 Tabla Cruzada. 165
Figura Nº 5-63. Fragmento del código del método exportar_pdf. 166
Figura Nº 5-64. Vista paso 4 de un reporte Regular. 170
Figura Nº 5-65. Vista paso 6 de un reporte Regular. 171
Figura Nº 5-66. Fragmento del código del método agregar_restriccion. 171
Figura Nº 5-67. Fragmento del código del método agregar_restriccion. 172
Figura Nº 5-68. Fragmento del código del método paso_6. 172
Figura Nº 5-69. Interfaz del paso 6 del wizard. 176
Figura N° 5-70. Ayuda del paso 3 del wizard. 177
Figura N° 5-71. Código del método ayuda. 178
Figura Nº B-1. Registro de datos del reporte de la Prueba 1. 198
Figura Nº B-2. Registro de tipo del reporte de la Prueba 1. 198

Figura Nº B-3. Selección de campos del reporte de la Prueba 1. 199
Figura Nº B-4. Definición de restricciones del reporte de la Prueba 1. 200
Figura Nº B-5. Selección de campos de ordenamiento del reporte de la Prueba 1. 201
Figura Nº B-6. Resultados del reporte de la Prueba 1. 201
Figura Nº B-7. Resultados del reporte de la Prueba 1 en formato Excel. 202
Figura Nº B-8. Selección de campos del reporte de la Prueba 2. 204
Figura Nº B-9. Definición de restricciones del reporte de la Prueba 2. 205
Figura Nº B-10. Definición de ordenamiento del reporte de la Prueba 2. 206
Figura Nº B-11. Registro de valores de las restricciones variables del reporte de la Prueba 2. 206
Figura Nº B-12. Resultado del reporte de la Prueba 2. 207
Figura Nº B-13. Resultado del reporte de la Prueba 2 en formato PDF. 208
Figura Nº B-14. Registro de la consulta del reporte de la Prueba 3. 209
Figura Nº B-15. Resultado del reporte de la Prueba 3. 210
Figura Nº B-16. Resultado del reporte de la Prueba 3 en formato CSV. 211
Figura Nº B-17. Selección de campos del reporte de la Prueba 4. 214
Figura Nº B-18. Definición de restricciones del reporte de la Prueba 4. 215
Figura Nº B-19. Registro de valores de las restricciones variables del reporte de la Prueba 4. 216
Figura Nº B-20. Resultados del reporte de la Prueba 4. 216
Figura Nº B-21. Resultados del reporte de la Prueba 4 en formato CSV. 217
Figura Nº B-22. Selección de campos del reporte de la Prueba 5. 219
Figura Nº B-23. Definición de restricción del reporte de la Prueba 5. 220
Figura Nº B-24. Resultados del reporte de la Prueba 5. 221
Figura Nº B-25. Resultado del reporte de la Prueba 5 en formato Excel. 222
Figura Nº B-26. Selección de campos del reporte de la Prueba 6. 224
Figura Nº B-27. Definición de valores de las restricciones del reporte de la Prueba 6. 225
Figura Nº B-28. Resultados del reporte de la Prueba 6. 225
Figura Nº B-29. Resultado del reporte de la Prueba 6 en formato PDF. 226

ÍNDICE DE TABLAS

Tabla Nº 2-1. Formatos soportados por Crystal Reports XI 40
Tabla Nº 2-2. Formatos soportados por Navicat 50
Tabla Nº 2-3. Formatos soportados por SQL Manager for MySQL 2007 56
Tabla Nº 2-4. Comparación de las herramientas 59
Tabla Nº 4-1. Iteraciones e historias de usuario 86
Tabla Nº 4-2. Plantilla para la representación de las historias de usuario 87
Tabla Nº 4-3. Roles existentes durante el desarrollo 88
Tabla Nº 5-1. Historias de usuario. Iteración 0 95
Tabla Nº 5-2. Historias de usuario. Iteración 1 99
Tabla Nº 5-3: Historias de usuario. Iteración 2 103
Tabla Nº 5-4: Historias de usuario. Iteración 3 109
Tabla Nº 5-5: Historias de usuario. Iteración 4 113
Tabla Nº 5-6: Historias de usuario. Iteración 5 121
Tabla Nº 5-7: Historias de usuario. Iteración 6 127
Tabla Nº 5-8: Historias de usuario. Iteración 7 135
Tabla Nº 5-9: Historias de usuario. Iteración 8 143
Tabla Nº 5-10: Historias de usuario. Iteración 9 151
Tabla Nº 5-11: Historias de usuario. Iteración 10 159
Tabla Nº 5-12: Historias de usuario. Iteración 11 163
Tabla Nº 5-13: Historias de usuario. Iteración 12 169
Tabla Nº 5-14: Historias de usuario. Iteración 13 175

INTRODUCCIÓN

La información es un elemento de vital importancia para cualquier empresa y organización ya que influye en la forma en la que éstas funcionan, por tanto en todo sistema de información es fundamental contar con una herramienta complementaria cuya función sea la de crear reportes a partir de los datos almacenados.

El objetivo que buscan cumplir los generadores de reportes es proveer a los usuarios, de una manera concisa y en un formato de fácil lectura, la información contenida en los repositorios de datos consultados. De manera que la creación de los mismos se debe realizar a través de herramientas que agilicen y faciliten su elaboración, y los formatos en que estos son generados deben permitir al usuario la comprensión y manejo de los datos contenidos en él.

En la actualidad existen generadores de reportes comerciales como Crystal Reports. El problema con este y otros generadores es que su costo es demasiado elevado, y como consecuencia no pueden ser utilizados por muchas organizaciones que necesitan una herramienta de este tipo. Además estas trabajan directamente con el modelo de datos, manejando en ciertas ocasiones gran cantidad de tablas y relaciones entre ellas.

El objetivo de este Trabajo Especial de Grado es construir un módulo para el sistema CONEST (el cual es una aplicación Web que automatiza los procesos de la gestión académica de la División de Control de Estudios) que permita la generación y administración de reportes de los datos almacenados en el repositorio de datos del sistema. A este módulo se le denomina CONEST Reportes y para su desarrollo se eligió la metodología XP.

Para cumplir el objetivo, se realizó un análisis del modelo de datos de CONEST y se construyeron los esquemas que representan los diferentes elementos que integran el modelo de datos académico, sobre el cual trabaja CONEST Reportes.

También se realizó un estudio para conocer las principales características de algunos reporteadores comerciales como: Crystal Reports XI, Navicat 8.0 for MySQL y EMS SQL Manager for MySQL 2007. El conocer las funciones que realizan estas herramientas ayudó en la definición de las características funcionales de CONEST Reportes.

De tal forma, este documento está estructurado de la siguiente manera:

Parte I: Propuesta de Trabajo Especial de Grado. Se presenta la propuesta de Trabajo Especial de Grado, en base a la investigación realizada y al conocimiento de una necesidad puntual dentro de la División de Control de Estudios de la Facultad de Ciencias. Se construye el objetivo general y los objetivos específicos del trabajo a realizar, así como también se destaca el proceso de desarrollo a utilizar y la plataforma tecnológica que servirá de base para la implementación de la solución.

Parte II: Marco Conceptual. Se estudian las características generales de herramientas generadoras de reportes, se hace énfasis especial en algunas herramientas comerciales existentes hoy en día para tal fin.

Luego en esta parte se describe cómo se realiza actualmente el proceso de generación de reportes y se analizan los inconvenientes asociados a dicho proceso dentro de la División de Control de Estudios de la Facultad de Ciencias, adicionalmente se presentan diversas clasificaciones de los reportes que se generan. Por último se hace referencia al manejador de Bases de Datos MySQL, empleado en el sistema CONEST, centrando el análisis en una de las características del manejador, el manejo de vistas.

Parte III: Marco Aplicativo. Donde se describen los aspectos más relevantes del proceso de desarrollo Programación Extrema (XP). Luego se explica de qué manera se llevó a cabo la construcción del generador de reportes, a través del uso de la metodología XP.

Para terminar se presentan las conclusiones, recomendaciones y referencias bibliográficas consultadas durante esta investigación.

PARTE I

PROPUESTA DE TRABAJO ESPECIAL DE GRADO

CAPÍTULO 1

PROPUESTA DE TRABAJO ESPECIAL DE GRADO

Se presenta a continuación la propuesta del Trabajo Especial de Grado, la cual habla sobre la problemática existente al momento de elaborar reportes en la División de Control de Estudios de la Facultad de Ciencias y se plantea la creación de un módulo de generación de reportes que facilite la elaboración y obtención de reportes.

1.1 Descripción del Problema

Actualmente la Facultad de Ciencias de la Universidad Central de Venezuela dispone de una aplicación Web que automatiza los procesos de la División de Control de Estudios denominada CONEST. Este sistema almacena toda la información correspondiente a la gestión académica de la Facultad, incluyendo información de los estudiantes, su historial académico, así como información de las materias, los docentes, las aulas, etc.

Para los distintos procesos que lleva a cabo la División de Control de Estudios y las diversas solicitudes que reciben de los departamentos y docentes que laboran en la Facultad, es necesario emitir una serie de reportes, que incluyen generalmente datos de los estudiantes, de las materias y otros reportes relacionados directamente con operaciones del sistema como tal.

En la actualidad, cuando en la División de Control de Estudios se reciben las solicitudes de reportes, se lleva a cabo un proceso de análisis de los datos solicitados y se delega sobre varios integrantes la responsabilidad de emitir los reportes de acuerdo a su complejidad.

Para lograr esto, los encargados de esta función de elaborar reportes, se han visto en la necesidad de aprender el lenguaje de consultas SQL, dado que los datos se encuentran almacenados en un repositorio de datos relacional, cuyo modelo lógico de datos es de un tamaño considerablemente grande. Para el momento de realizar esta investigación el modelo de datos tenía aproximadamente 140 tablas, lo cual añade un grado de dificultad mayor y exigen conocimientos avanzados en el área de Base de Datos para poder realizar los reportes.

Tomando esto en cuenta, cada integrante del grupo de trabajo utiliza un grupo de herramientas especializadas que permiten hacer consultas en el lenguaje SQL. Una vez que se tiene la consulta elaborada, esta es ejecutada y los resultados de la misma son copiados en

aplicaciones de oficina en las cuales se les da formato y en algunos casos se generan gráficos, para ser presentados a los solicitantes del reporte.

Los resultados de los reportes que son generados se emiten sin ningún formato predefinido y por ello necesitan ser manejados y formateados manualmente en una aplicación de oficina. Este último paso es propenso a que se cometan errores y además se invierte mucho tiempo dándole formato a los datos, proceso que se puede automatizar.

Debido a esta complejidad del modelo de datos y al problema del manejo de los resultados de las consultas, ha surgido la necesidad de automatizar este proceso de generación de reportes con el fin de evitar que se cometan errores y que el tiempo empleado en su elaboración sea mucho menor.

Como propuesta del Trabajo Especial de Grado se plantea desarrollar un generador de reportes que permita realizar y emitir reportes con los datos almacenados en el repositorio de datos del sistema CONEST a través de la Web, basándose en la problemática y estudios realizados en el Trabajo de Seminario.

En este trabajo se propone implementar una solución tomando en cuenta algunas características de generadores de reportes comerciales. De estas herramientas resaltan un conjunto de características que serán tomadas en cuenta para el desarrollo de la propuesta. A continuación se explican algunas de ellas:

- La generación de los reportes se simplifica mediante el establecimiento de secuencias de pasos (wizard), ya que estas guían al usuario a organizar la elaboración de reportes. En el Trabajo de Seminario se evaluaron varias herramientas, de las cuales se tomará como guía el proceso de creación de reportes en Crystal Reports XI, debido a que integra la generación de la consulta, la selección de campos y la selección de características de formato básicas del reporte, todo en una misma secuencia de pasos.

- La posibilidad de poder definir una consulta en lenguaje SQL y utilizarla como base para la elaboración del reporte. El usuario puede tener ya elaborada la consulta y lo que desea es simplemente tomar los resultados de esta consulta y mostrarla en un formato.

- El usuario puede guardar los reportes para posterior obtención de los resultados.

- Se puede elaborar un reporte a partir de otro ya elaborado.

La propuesta tiene las siguientes características:

a. Los usuarios del sistema CONEST, a través del módulo de generación de reportes, pueden elegir entre crear un nuevo reporte (ver punto 1 en la Figura Nº 1-1) o seleccionar de un listado uno de los reportes previamente elaborados en el sistema (ver punto 5 en la Figura Nº 1-1).

b. En caso de haber elegido crear un nuevo reporte, el usuario podrá construir la consulta asociada al reporte mediante una secuencia de pasos definidos que le permitirán consultar los datos de las entidades académicas identificadas en el Trabajo de Seminario, de forma tal que simplifique la elaboración de estos reportes. El usuario también podrá introducir una consulta, previamente elaborada, en lenguaje SQL para obtener unos datos particulares del repositorio de datos.

c. En caso de haber seleccionado un reporte previamente creado, el usuario podrá obtener el resultado del reporte o modificarlo.

d. Una vez que el usuario haya creado una consulta o haya seleccionado un reporte, podrá ejecutar dicha consulta (ver punto 2 en la Figura Nº 1-1) y podrá ver una vista preliminar con los resultados del reporte en formato HTML (ver punto 3 en la Figura Nº 1-1).

e. Los usuarios podrán exportar los reportes a cualquiera de los formatos soportados por el sistema (PDF, EXCEL, CSV) (ver punto 4 en la Figura Nº 1-1).

f. Los usuarios podrán guardar una copia de un reporte ya existente como un nuevo reporte, siendo este agregado al listado para posibles ejecuciones en próximas ocasiones.

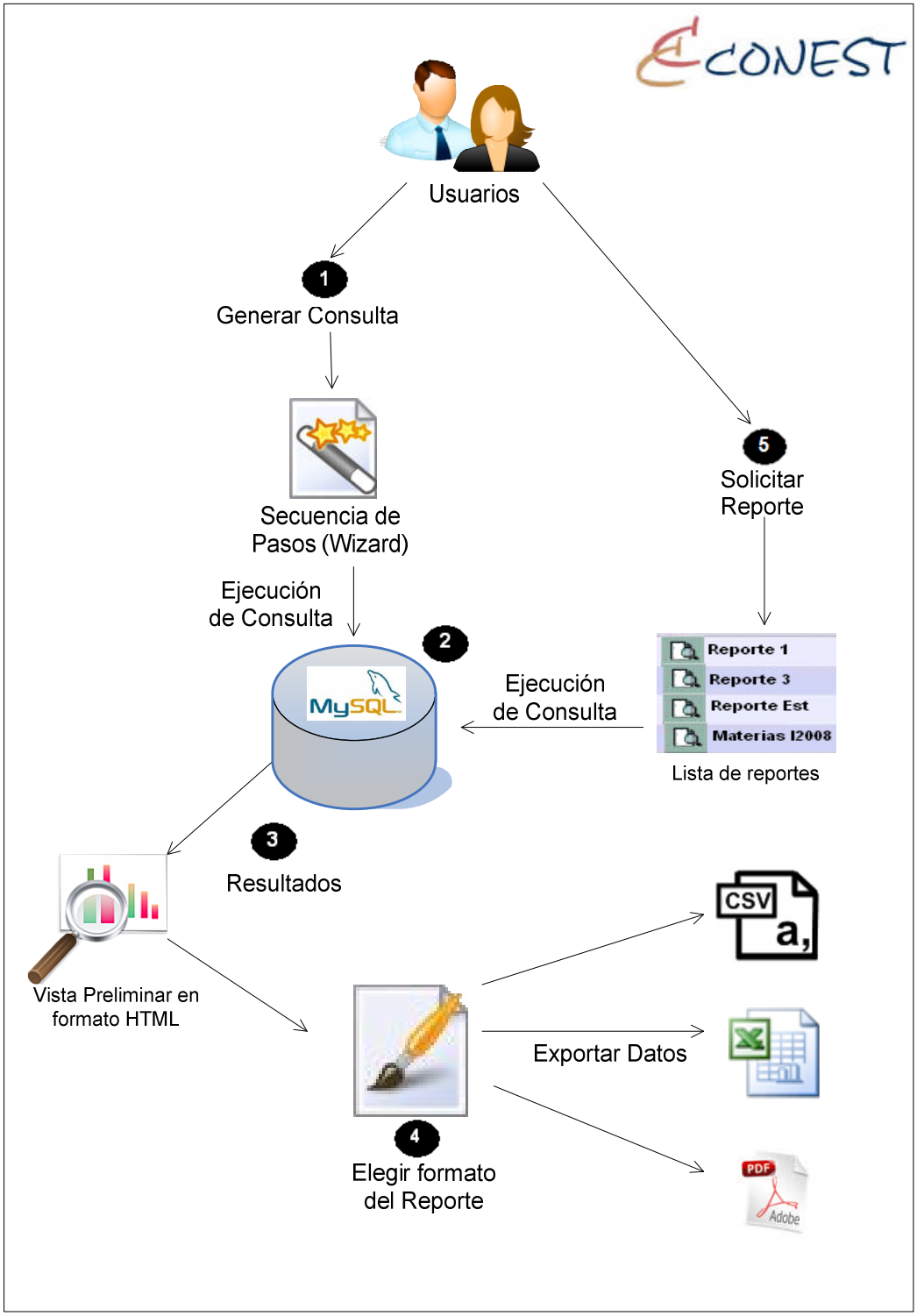


Figura Nº 1-1. Propuesta.

Tras la evaluación de las herramientas probadas en el Trabajo de Seminario, se tiene como práctica común en ellas la división del reporte en varias partes o zonas. La propuesta se basa en plantear tres partes básicas en todo reporte:

- **Encabezado:** Parte superior del reporte, que se mantiene en todas las páginas, en caso de tener más de una página el reporte. No contiene datos directamente relacionados con el reporte y sirve como un elemento de presentación. En ocasiones se tienen elementos variables como la fecha y generalmente se incluye el título del reporte.

- **Datos:** Parte central del reporte que contiene los datos.

- **Pie:** Parte inferior del reporte, que se mantiene en todas las páginas, en caso de tener más de una página el reporte. No contiene datos directamente relacionados con el reporte y sirve como un elemento de presentación. Generalmente incluye el número de página.

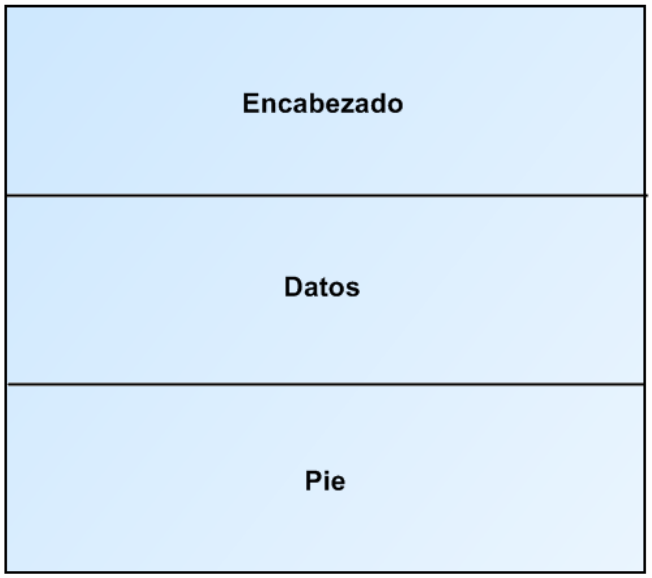


Figura Nº 1-2. Partes o zonas de un Reporte.

1.2 Objetivo General

Desarrollar un módulo de CONEST que permita la generación y administración de reportes de la gestión académica de la Facultad de Ciencias.

1.3 Objetivos Específicos

a. Estudio minucioso de la plataforma tecnológica y del modelo de datos existente empleado por CONEST.
b. Adaptar la metodología XP para el desarrollo de una Aplicación Web que permita la generación y administración de reportes.
c. Realizar el diseño físico y lógico de la Base de Datos que permita almacenar la información relacionada a la problemática planteada y la administración de los reportes.
d. Elaborar el diseño de las interfaces gráficas de usuario (GUI).
e. Diseñar y desarrollar las secuencias de pasos (wizards) que permitan la elaboración de reportes con los datos de las entidades identificadas.
f. Someter a pruebas rigurosas la aplicación Web.
g. Integrar con el sistema CONEST en producción.

1.4 Proceso de Desarrollo

El proceso de desarrollo a utilizar será Programación Extrema (Extreme Programming) o XP el cual es una metodología ágil de desarrollo de software que reduce la complejidad del mismo por medio del trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción. [Márquez, 2008].

1.5 Tecnologías y Plataformas

Durante el desarrollo se utilizarán las mismas tecnologías que utiliza CONEST ya que el repositorio y sistema generador de reportes estará integrado con éste y las tecnologías de CONEST han demostrado ser una implementación exitosa. Las tecnologías usadas por CONEST son de aplicaciones Web y son nombradas a continuación:

- Lenguaje de programación Ruby 1.8.6.
- Framework Web Rails 1.2.6.
- Sistema manejador de base de datos MySQL Server 5.0.

Las tecnologías propuestas para el módulo de generación de reportes son:

- Librería PDF::WRITTER, la cual va a permitir obtener los resultados del reporte en formato PDF.

- Librería Faster CSV, permitirá obtener los resultados del reporte en formato CSV.
- Librería Ruby Spreadsheet, para obtener los resultados del reporte en formato EXCEL.
- Librería REXML, para manejar los documentos XML asociados a cada reporte.

Para hacer uso de la aplicación sólo es necesario disponer de un navegador Web sobre cualquier plataforma.

PARTE II

MARCO CONCEPTUAL

CAPÍTULO 2

HERRAMIENTAS PARA ELABORAR REPORTES

Este capítulo trata sobre las herramientas que permiten la generación de reportes. Se hará un estudio de algunas de las características generales de este tipo de aplicaciones y se brindará un pequeño detalle de algunas que existen actualmente en el mercado. Al final de este capítulo se analizará un aspecto común en las aplicaciones estudiadas, que servirá como base para el diseño de la interfaz de usuario de la herramienta desarrollada en este Trabajo Especial de Grado.

2.1 Los reportes y las herramientas para elaborarlos

Un reporte puede ser visto como una representación de los datos y sus relaciones, tal y como aparecen en la base de datos. En el contexto de las Bases de Datos relacionales, un reporte muestra una lista de campos y registros, que han sido seleccionados de tablas y sus relaciones, mediante el uso de del lenguaje de consulta SQL.

Un reporte debe presentar la información que es obtenida a partir de una consulta de una forma atractiva y fácil de leer.

Los sistemas generadores de reportes, tienen en las bases de datos su principal fuente de alimentación, y han brindado al usuario final (entendido éste como cualquier persona que requiera un reporte), la posibilidad de consultar y publicar lo que las bases de datos poseen.

Un generador de reportes simplemente es una herramienta que automatiza la estructura y la visualización de la información que consultamos de una base de datos determinada.

La generación de reportes es fundamental, ya que los reportes permiten reflejar, en un formato comprensible, el comportamiento de los diferentes componentes que integran cualquier organización. Esto es importante para el reconocimiento de las diversas problemáticas que existen en la misma y en esa medida apoyar en la correcta toma de decisiones que resuelvan dichas problemáticas.

La limitante que siempre ha existido en este sentido es que generar un reporte implica contar con algunas habilidades técnicas, relacionadas con el manejo de datos en las bases de datos y las herramientas de software.

A continuación se presenta un resumen de las principales características de algunos de los generadores de reportes comerciales estudiados.

2.2 Crystal Reports XI Professional

2.2.1 Características

a. **Tipo de Licencia:** Propietario.
b. **Empresa desarrolladora:** Business Objects.
c. **Año de publicación:** 2004.
d. **Versión utilizada:** 11.0.0.895
e. **Características generales** [Mañón, 2004]

Este generador de reportes proporciona herramientas para la creación de reportes y consultas avanzadas tanto para el usuario experimentado como para el principiante, permitiendo diseñar informes y exportar datos de manera sencilla. Mediante el uso de este generador de reportes se puede seleccionar, analizar, resumir y presentar información en casi cualquier forma que se desee. Su potencialidad radica en la posibilidad de enlazarse a diversas fuentes de datos, desde hojas de cálculo electrónicas hasta manejadores de bases de datos relacionales.

Entre las funcionalidades principales de esta herramienta se encuentran que permite integrar sub-reportes a un reporte principal, permite presentar información resumida en una sola celda en lugar de presentarla en columnas, le da al usuario más control sobre el diseño de los reportes, permite el manejo de gráficos, fórmulas y subtotales, entre otros.

Esta herramienta requiere que el usuario posea conocimientos sobre la lógica de cómo los datos se almacenan en las bases de datos relacionales y del significado de dichos datos y las relaciones entre ellos.

2.2.2 Proceso de generación de reportes [Márquez, 2008]

Al momento de crear un reporte, la herramienta ofrece tres opciones:

- **Mediante Secuencia de pasos (Wizards):** Esta opción guiará al usuario paso por paso durante la creación del reporte. Existen varios tipos: reporte de tabla cruzada (ver Figura Nº 2-1), reporte estándar (ver Figura Nº 2-4), etc.

- **Como un reporte en blanco (Blank Report):** Esta opción desplegará la interfaz de desarrollo de reportes en blanco para que el usuario cree el reporte sin asistencia (para usuarios expertos).

- **A partir de otro reporte (Open):** Esta opción permite crear un reporte a partir de otro ya existente, al seleccionar esta opción la herramienta solicita al usuario la localización del archivo del reporte (archivo de extensión .rpt) que quiere utilizar y creará una copia de este reporte.

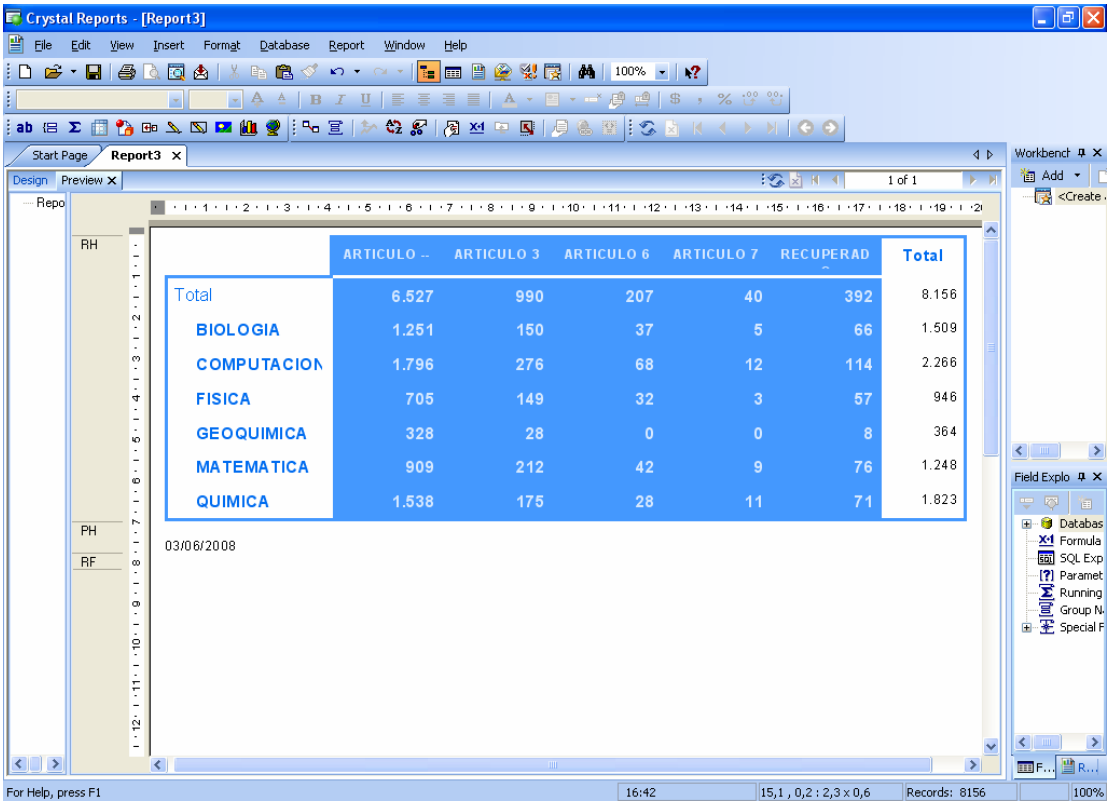


Figura Nº 2-1. Resultado de un reporte de tabla cruzada en Crystal Reports.

En las pruebas realizadas con el manejador de base de datos relacional MySQL, se tiene que Crystal Reports, como primer paso del Wizard de reporte estándar, permite seleccionar los datos que se desean en el reporte de dos formas:

- Mediante una consulta SQL elaborada por el usuario, a través de la opción de añadir comando.
- Seleccionando las tablas que contienen los datos que se van a utilizar en el reporte.

Como segundo paso, Crystal Reports permite seleccionar los campos a incluir en el reporte. Si se elige la opción de la consulta se obtienen una serie de campos sobre los cuales se

pueden seleccionar los que se requieren que efectivamente aparezcan en el reporte (ver Figura Nº 2-2).

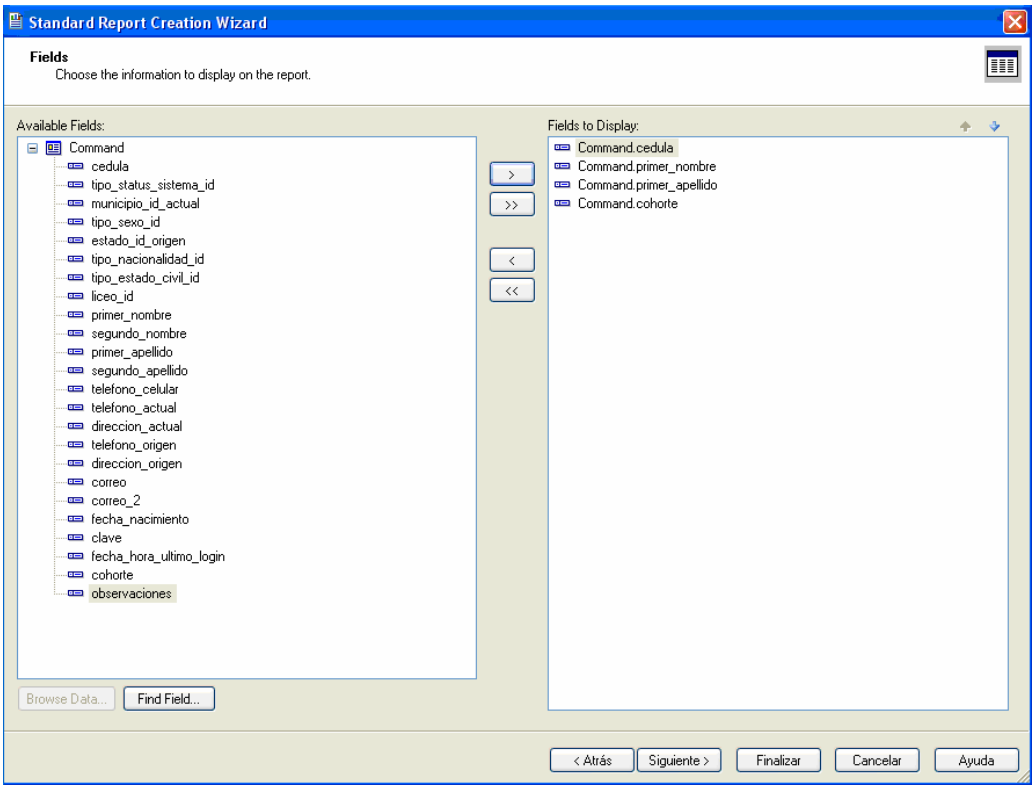


Figura Nº 2-2. Selección de campos en Crystal Reports.

Una vez seleccionados los campos, el siguiente paso le permite al usuario definir grupos para reunir los datos de acuerdo a un criterio. En el siguiente paso, el usuario puede definir funciones de conteo que serán evaluadas más adelante.

Luego se le permite al usuario poder definir restricciones o condiciones sobre algunos de los campos que se mostraran en el reporte o restricciones adicionales sobre los campos de la consulta definida en un principio.

Como último paso para generar el reporte, la herramienta le provee al usuario un conjunto de diseños predefinidos para el reporte que acaba de elaborar, y de los cuales se le permite seleccionar un diseño para visualizar el reporte (ver Figura Nº 2-3).

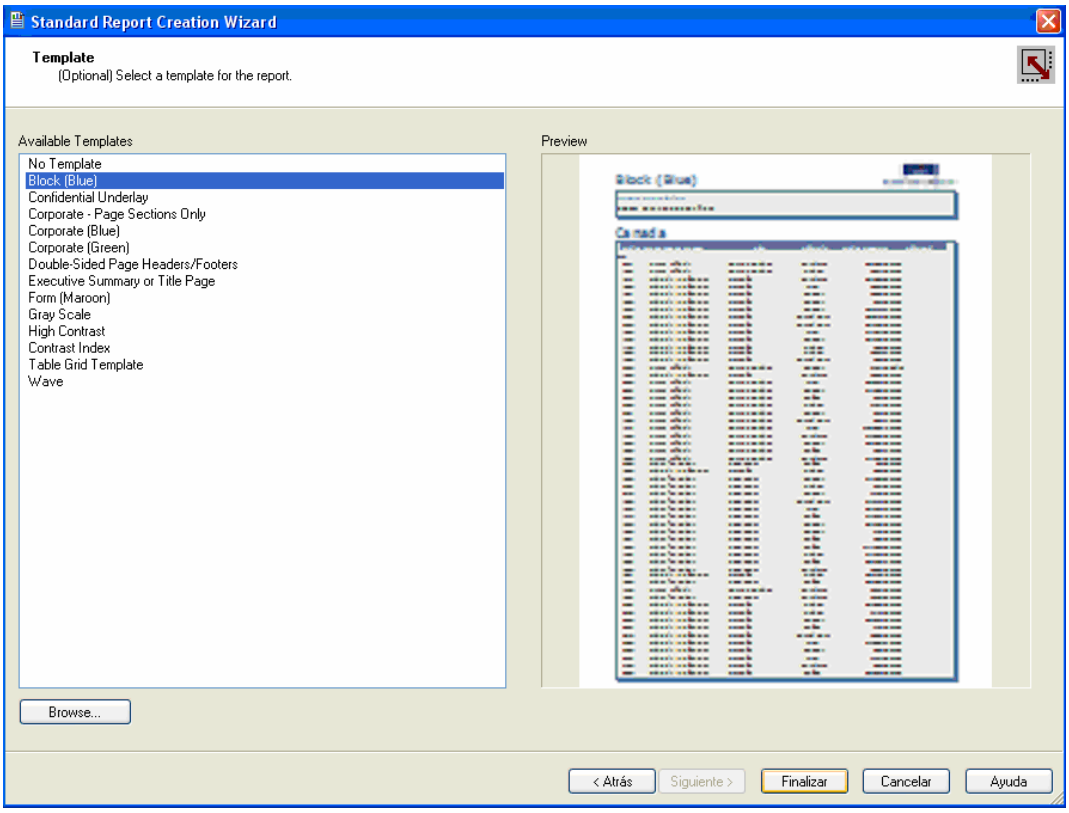


Figura Nº 2-3. Selección de diseño en Crystal Reports.

Posteriormente se muestra al usuario los resultados del reporte en una vista preliminar que le permite editar manualmente el formato del mismo (ver Figura Nº 2-4).

Una vez que el reporte ha sido elaborado, se puede editar el diseño del mismo. Para el manejo de las características de diseño del reporte, es recomendable que el usuario esté familiarizado con herramientas de oficina, como Word y Power Point del paquete Microsoft Office, ya que muchas características se asemejan en su manejo a elementos dentro de estas herramientas.

La herramienta permite incluir encabezado, pie de página, editar colores, el tipo y tamaño de la fuente, definir tamaño de los reportes, márgenes y ubicación de los elementos en el reporte.

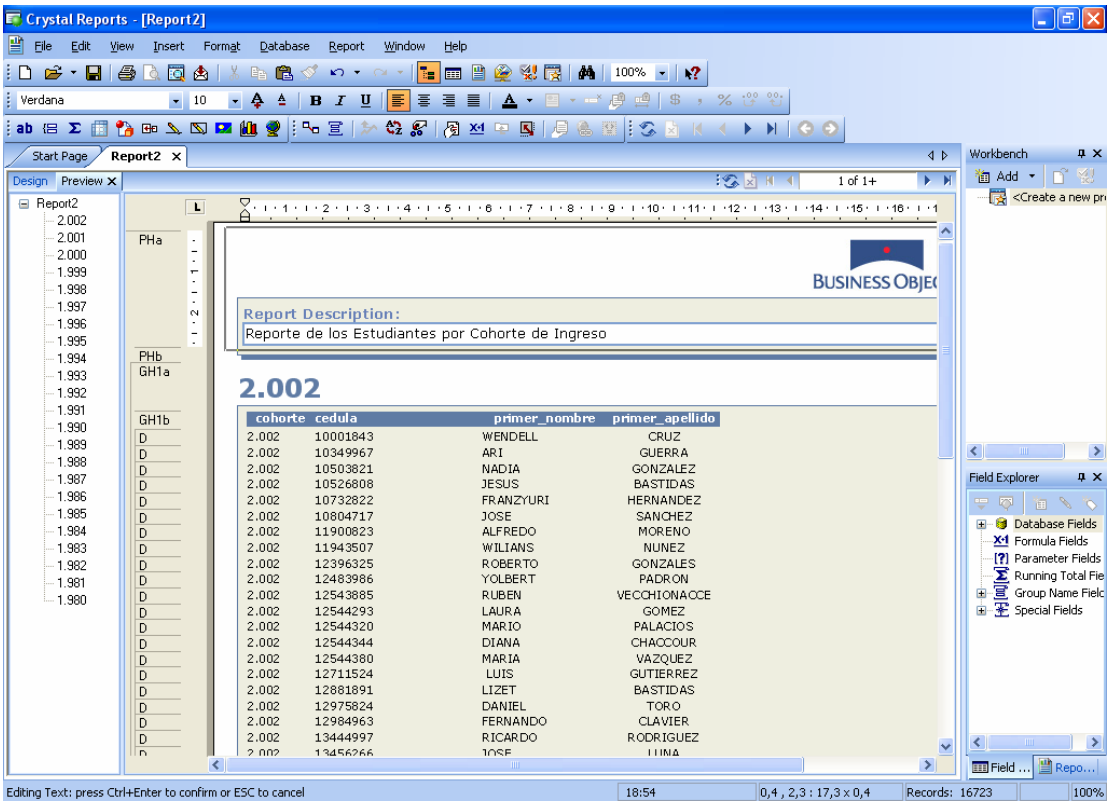


Figura Nº 2-4. Resultado de un reporte estándar en Crystal Reports.

La otra forma de hacer los reportes es seleccionando las tablas que contienen a los datos que se van a utilizar en el reporte. Cuando se seleccionan las tablas el programa busca relacionarlas y muestra un gráfico para reflejar las relaciones entre ellas (ver Figura Nº 2-5)

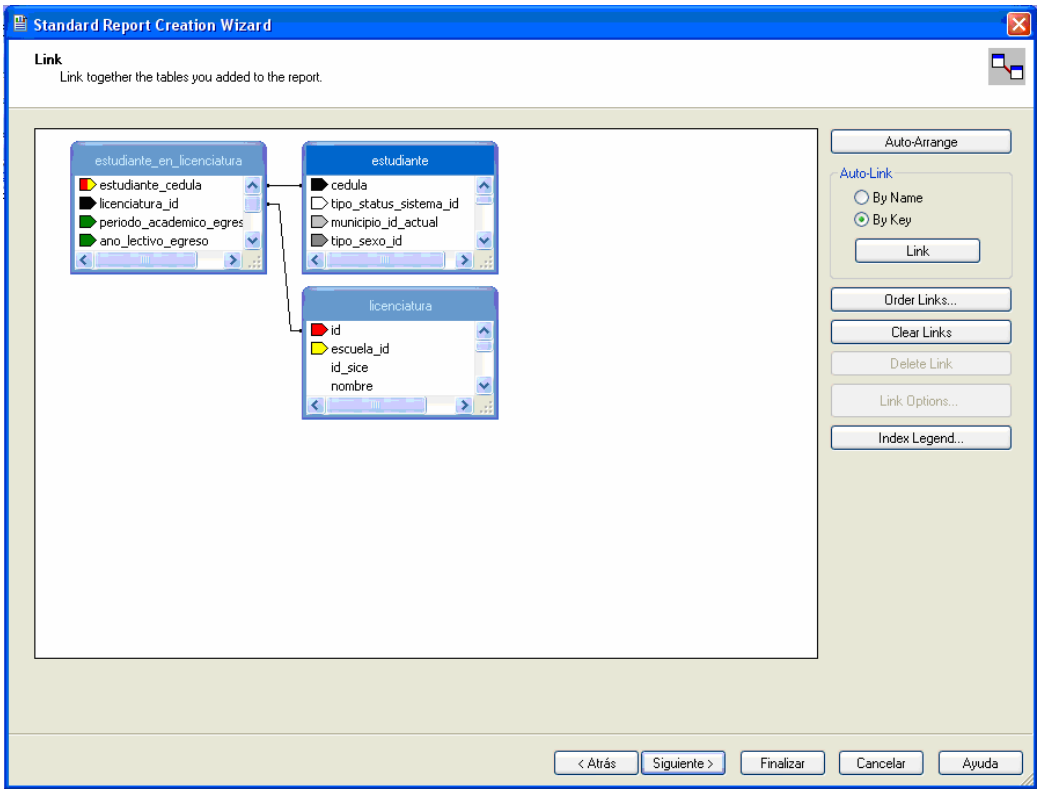


Figura Nº 2-5. Relaciones entre las tablas en Crystal Reports.

Luego el proceso es exactamente el mismo, seleccionar los campos específicos que se desean mostrar, definir grupos y así sucesivamente.

En cambio si se elige crear un reporte de tipo de tabla cruzada, el proceso es básicamente el mismo sólo cambia radicalmente al momento de seleccionar los campos a incluir en el reporte (ver Figura Nº 2-6).

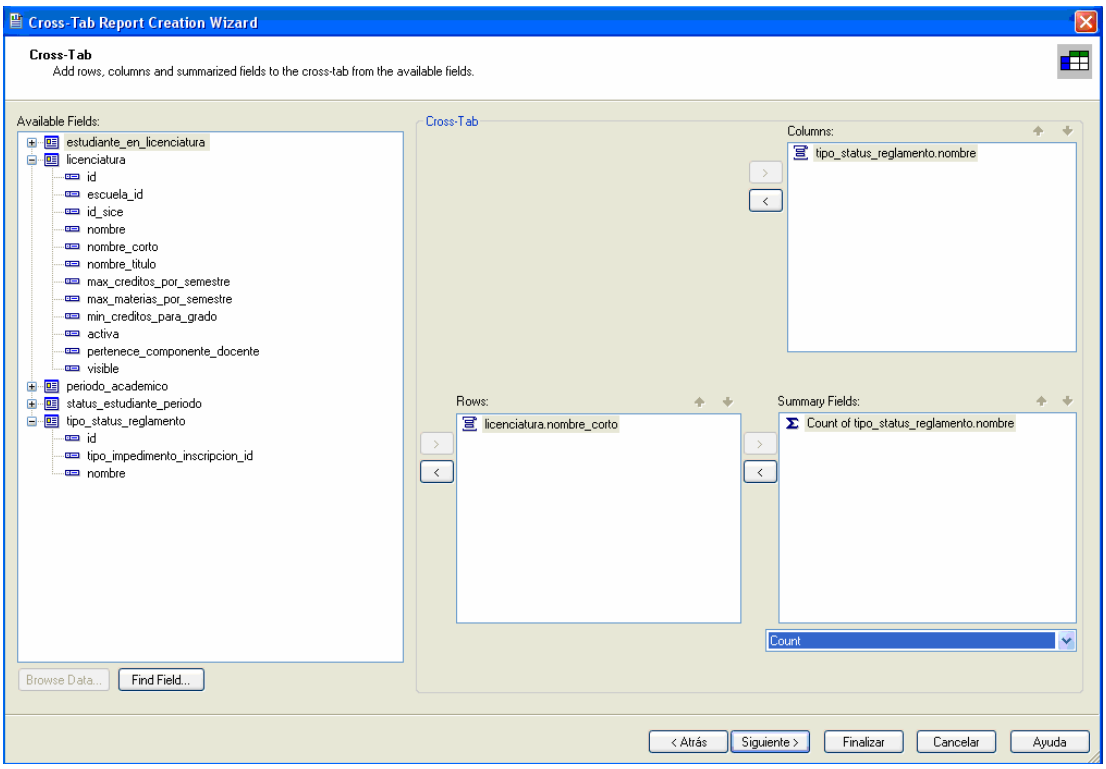


Figura Nº 2-6. Disposición de los campos en la tabla del reporte de Tabla Cruzada de Crystal Reports.

Una vez que se tiene el reporte generado puede ser guardado o exportado a uno de los múltiples formatos soportados.

Crystal Reports permite generar el reporte en múltiples formatos, algunos de ellos buscan mantener el diseño, colores, y formas que fueron utilizadas en el diseño del reporte y otros sólo almacenan los datos (ver Tabla Nº 2-1).

Formato	Características
PDF	Se genera el reporte manteniendo todas las características del diseño elaborado en la aplicación. Permite ver gráficos, imágenes, etc.
HTML (Hyper Text Markup Language) Ver. 3.2 y 4.0	Se genera el reporte manteniendo todas las características del diseño elaborado en la aplicación. Permite ver gráficos, imágenes, etc.

Excel (XLS)	Los datos se almacenan, siguiendo el formato de Excel, en las celdas manteniendo tal cual todas las características el diseño elaborado en la aplicación. Permite ver gráficos, imágenes, etc. Los gráficos no pueden ser editados, ya que son almacenados como imágenes.
Excel (Data Only)	Sólo se almacenan los datos en las celdas, no se toman en cuenta las características de formato definidas por el usuario. No se almacenan imágenes ni gráficos.
RTF (Rich Text Format)	Formato en el que el reporte mantiene todas las características de diseño elaboradas al generar el reporte. Este formato es útil si se desea que el reporte pueda ser visualizado mediante procesadores de texto como Word (paquete Office de Microsoft) o Writer (paquete OpenOffice de Sun Microsystems).
CSV (Comma Separated Values)	Este formato sólo almacena los datos que fueron seleccionados al generar el reporte, es decir, almacena campo a campo de los registros producto de la consulta realizada, separados por coma.
TTX (Tab Separated Text)	Este formato sólo almacena los datos que fueron seleccionados al generar el reporte, es decir, almacena campo a campo de los registros producto de la consulta realizada, separados por tabulador.
Texto sin formato (TXT)	Este formato sólo almacena los datos que fueron seleccionados al generar el reporte, es decir, almacena campo a campo de los registros producto de la consulta realizada, en formato de texto sin ninguna característica de formato.

Tabla Nº 2-1. Formatos soportados por Crystal Reports XI.

2.2.3 Evaluación de la Herramienta [Márquez, 2008]

Mediante las pruebas realizadas, se pueden definir las siguientes ventajas y desventajas de la herramienta.

- **Ventajas**
  - Amplia variedad de formatos en los que se puede generar un reporte, bien sea manteniendo o no el estilo y diseño elaborado por el usuario en la aplicación.
  - La herramienta es de fácil uso. De todas las herramientas probadas fue la que tuvo la curva de aprendizaje más corta, permitiendo obtener resultados elaborados sin necesidad de un entrenamiento exhaustivo en la herramienta.
  - Permite la generación de reportes de listados y de tabla cruzada mediante secuencias de pasos (wizards) especializados para cada caso.

- Permite obtener los datos desde múltiples fuentes de datos heterogéneas.
- Permite la generación de gráficos a partir de los datos consultados.
- Permite elaborar reportes a partir de la selección de tablas o a partir de una consulta escrita en lenguaje SQL.
- Al elaborar un reporte a partir de la selección de tablas, la herramienta ayuda al usuario en la obtención de los datos mediante la sugerencia de los enlaces entre las tablas seleccionadas.
- La herramienta permite la edición del formato del reporte de forma sencilla.

– **Desventajas**

- Es una herramienta propietario que tiene un costo elevado, alrededor de $500 la versión Crystal Reports XI Professional utilizada.
- Según la herramienta, los requisitos mínimos de la misma son bastante bajos, pero al realizar la sugerencia de los enlaces entre las tablas se demora mucho tiempo, en algunos casos varios minutos, aún contando con un equipo que supera por mucho los requisitos mínimos de rendimiento. Por lo cual exige de un procesador de alto rendimiento y de amplia capacidad de memoria.
- La conexión con el manejador de Base de Datos Relacional de MySQL no es directa a través de la herramienta, ya que necesita que se instale un software adicional en el equipo donde se instale la herramienta. Este software es el conector ODBC para MySQL que se consigue en la página oficial del manejador de base de datos.

2.3 Navicat 8

2.3.1 Características

a. **Tipo de Licencia:** Propietario.
b. **Empresa desarrolladora:** PremiumSoft.
c. **Año de publicación:** 2008.
d. **Versión utilizada:** 8.0.27.
e. **Características generales:** [Navicat, 2006]

Navicat MySQL es una herramienta que permite el desarrollo y administración de Bases de Datos MySQL que posee un componente que permite la generación de reportes. Se encuentra disponible para tres plataformas: Microsoft Windows, Mac OS X y Linux. Esta herramienta trabaja con todas las versiones de MySQL a partir de la versión 3.21 y permite el manejo de disparadores (Triggers), funciones, vistas, manejo de usuarios, procedimientos almacenados (Stored Procedure), entre otros.

Es una herramienta bastante intuitiva que permite crear, navegar sobre bases de datos y ejecutar consultas SQL de una forma sencilla, pero es lo suficientemente sofisticada para cumplir con los requerimientos para el manejo de la base de datos MySQL. Debido a esto, la herramienta requiere que el usuario posea conocimientos sobre la lógica de cómo los datos se almacenan en las bases de datos relacionales.

2.3.2 Proceso de generación de reportes [Márquez, 2008]

Al momento de crear un reporte, la herramienta ofrece tres opciones (ver Figura Nº 4-1):

- **Mediante Secuencia de pasos (Wizards):** Esta opción guiará al usuario paso por paso durante la creación del reporte. Existen varios tipos: reporte de tabla cruzada (ver Figura Nº 2-18), reporte de listado (ver Figura Nº 2-16), etc.

- **Como un reporte en blanco (Report):** Esta opción desplegará la interfaz de desarrollo de reportes en blanco para que el usuario cree el reporte sin asistencia (para usuarios expertos).

- **A partir de otro reporte (Open):** Esta opción permite crear un reporte a partir de otro ya existente, al seleccionar esta opción la herramienta solicita al usuario la localización del

archivo del reporte (archivo de extensión .rtm) que quiere utilizar y creará una copia de este reporte.

A través de esta herramienta lo primero que hay que elaborar es la consulta, de forma separada al reporte en sí.

Navicat dentro del componente de reportes provee dos formas de elaborar consultas a la Base de datos. Una de ellas a través de una secuencia de pasos (Query Wizard) y otra (Query Designer) en el que el usuario puede definir directamente la consulta SQL o diseñarla, sin seguir una secuencia predefinida de pasos, con las opciones que allí se le provee (ver Figura Nº 2-7). En caso de que se tenga una consulta SQL ya elaborada o el usuario sea experto en el desarrollo de consultas a la base de datos, se puede elegir la opción del Query Designer.

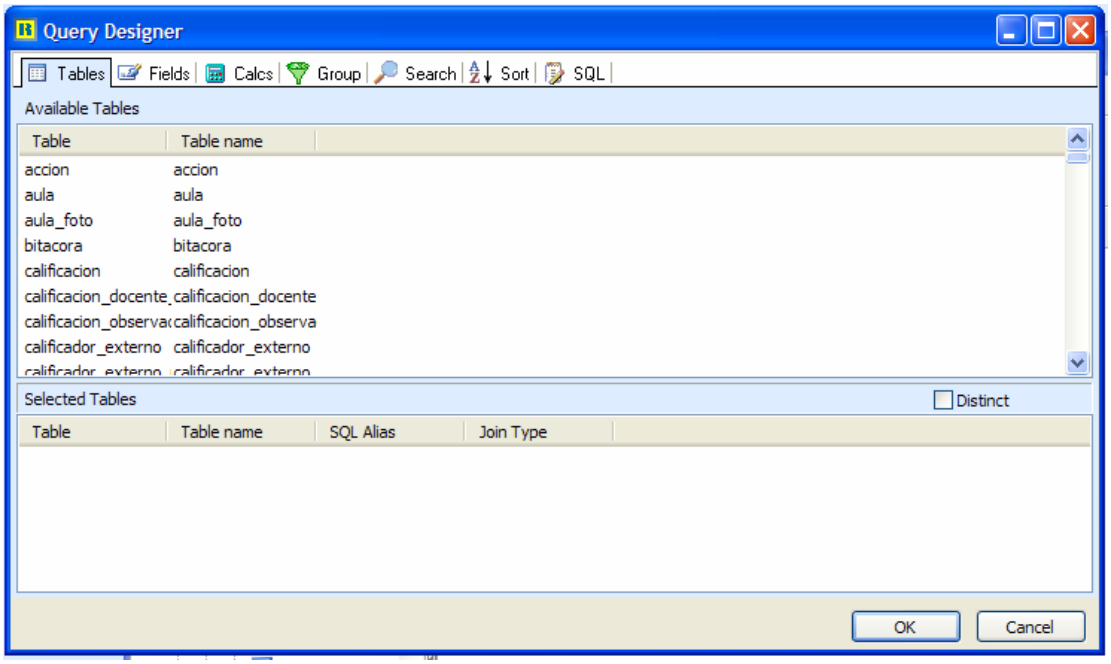


Figura Nº 2-7. Query Designer de Navicat.

Si se selecciona la opción del Query Wizard, ésta guía al usuario por una serie de pasos para elaborar la consulta. Lo primero que pide el Wizard son las tablas necesarias para realizar el Query (ver Figura Nº 2-8).

Figura Nº 2-8. Selección de tablas para el reporte de listado en Navicat.

En caso de seleccionar varias tablas, Navicat trata de relacionarlas por defecto por aquellos campos cuyos nombres sean iguales (ver Figura Nº 2-9). Si la relación no es a través de estos campos, el usuario puede eliminar esta relación y seleccionar manualmente los campos que deben relacionarse con mucha facilidad.

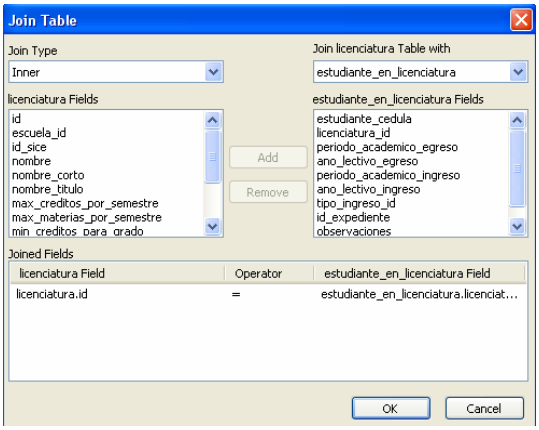


Figura Nº 2-9. Selección manual de los campos que deben relacionarse en el reporte de listado en Navicat.

Una vez seleccionadas todas las tablas de donde se van a obtener los datos, se deben seleccionar los campos a mostrar en el reporte (ver Figura Nº 2-10).

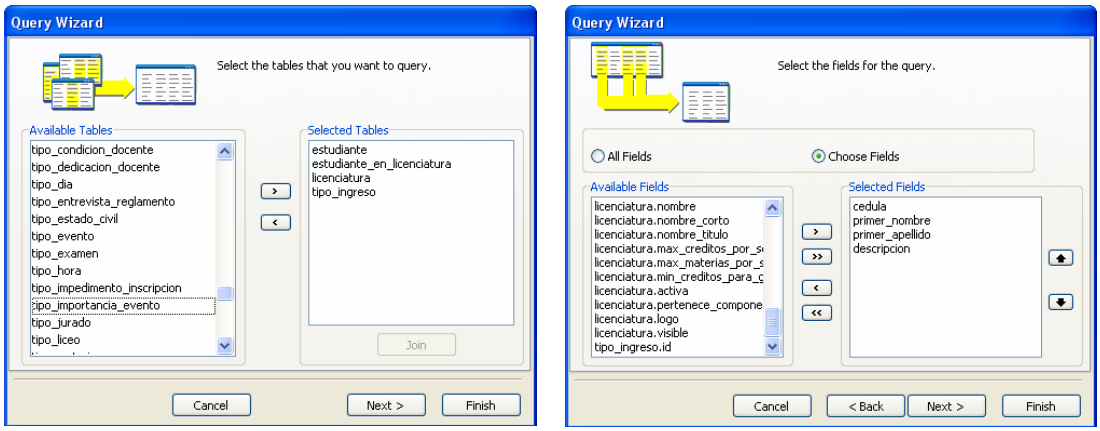


Figura Nº 2-10. Tablas y campos seleccionados en el reporte de listado en Navicat.

Luego los siguientes pasos permiten definir cálculos y grupos en los datos que se están consultando en la base de datos. Por último se definen las restricciones (ver Figura Nº 2-11).

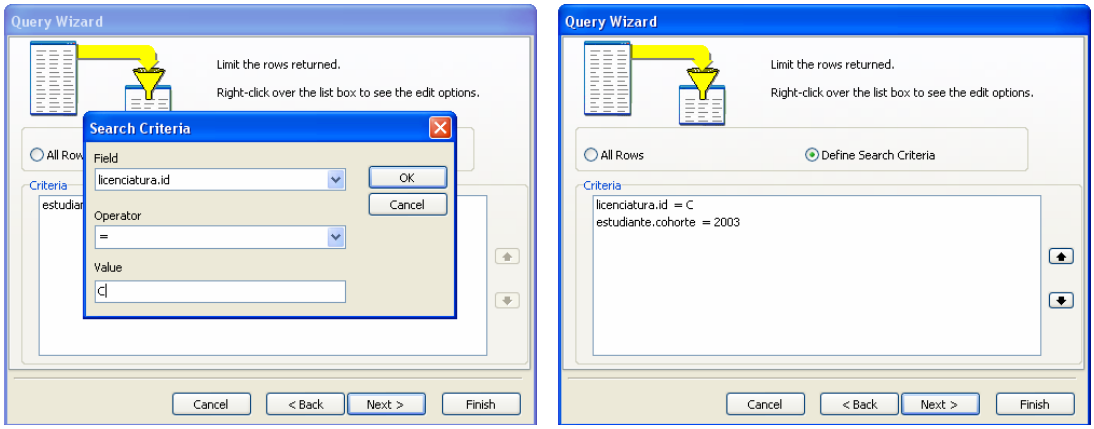


Figura Nº 2-11. Definición de restricciones en el reporte en Navicat.

En uno de los pasos del Wizard, se provee la posibilidad de ordenar los datos y para finalizar se coloca un nombre a la consulta que se acaba de elaborar. Luego se muestra una vista preliminar de los resultados obtenidos con la ejecución de la consulta (ver Figura Nº 2-12).

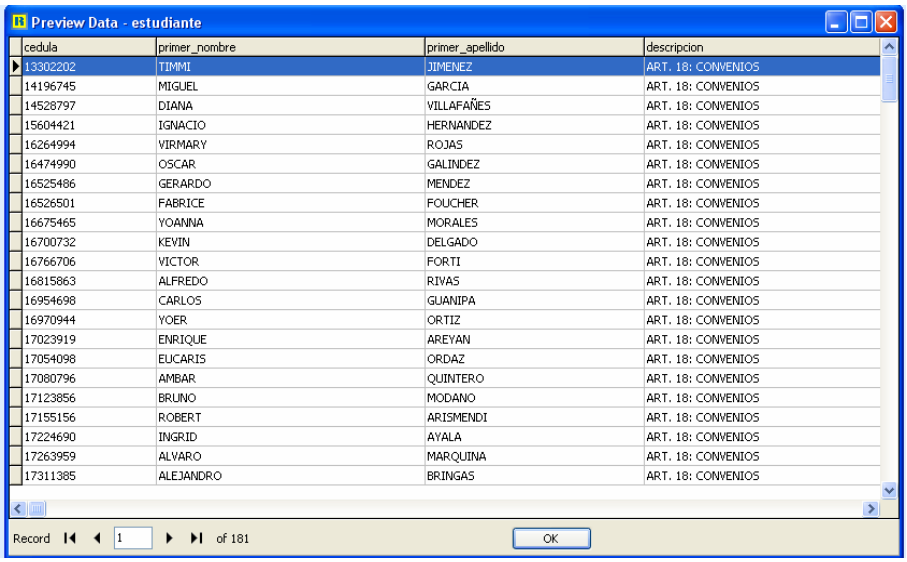


Figura Nº 2-12. Vista preliminar de los resultados en Navicat.

A continuación se procede a elaborar el reporte como tal. Para ello se debe seleccionar la pestaña ir a la sección de diseño del reporte, y luego elegir la opción de un nuevo del menú y elegir la el Wizard (Report Wizard) de reportes de listado (ver Figura Nº 2-12).

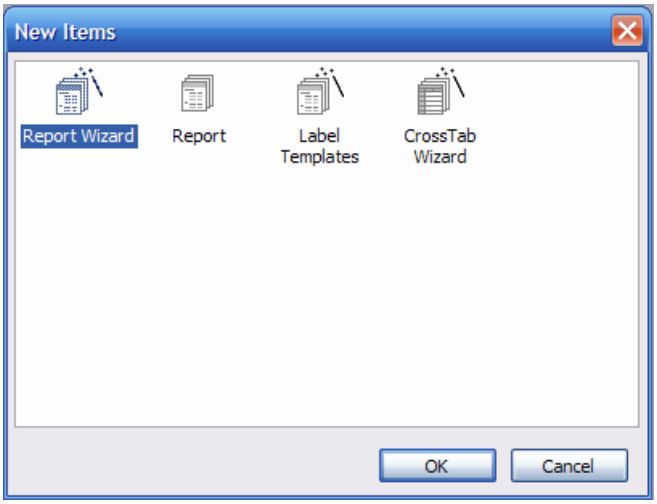


Figura Nº 2-13. Opciones para generar un reporte en Navicat.

Una vez elegida esta opción se puede visualizar que toma la consulta que se acaba de elaborar y esto permite seleccionar los campos que se van a mostrar en el reporte (ver Figura Nº 2-14).

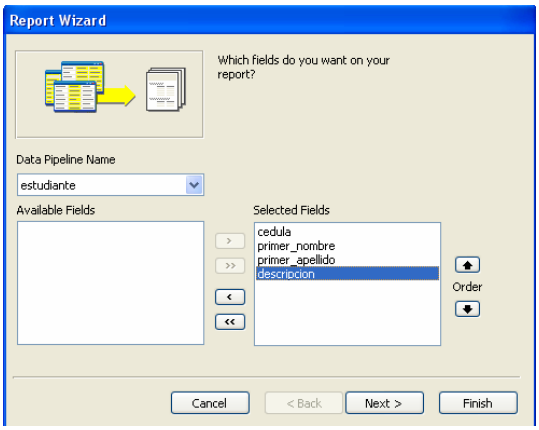


Figura Nº 2-14. Selección de campos en el reporte de listado en Navicat.

Posteriormente se puede dar formato al reporte y escoger un estilo, la aplicación viene con un conjunto de estilos predefinidos desde los cuales se puede elegir (ver Figura Nº 2-15).

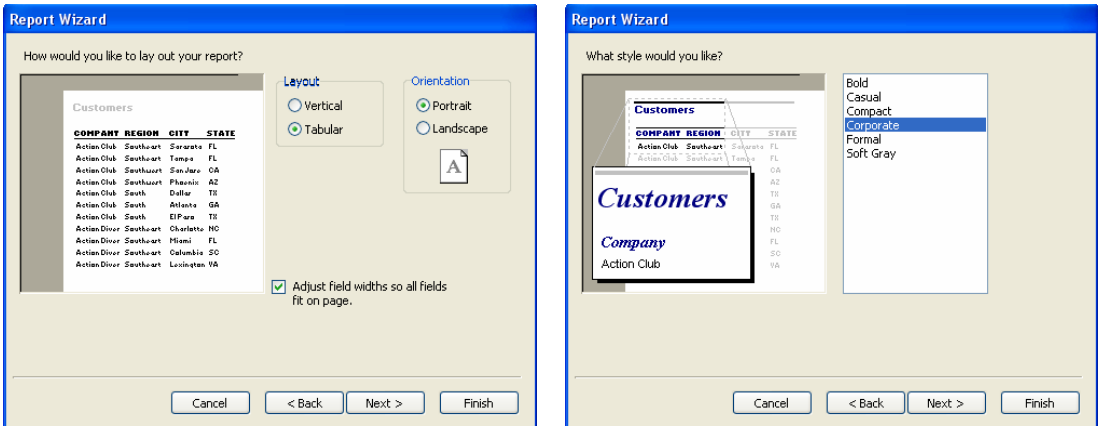


Figura Nº 2-15. Seleccionando un estilo en el reporte de listado en Navicat.

El reporte generado se puede visualizar mediante la opción de vista previa (Preview) como se puede apreciar en la Figura Nº 2-16.

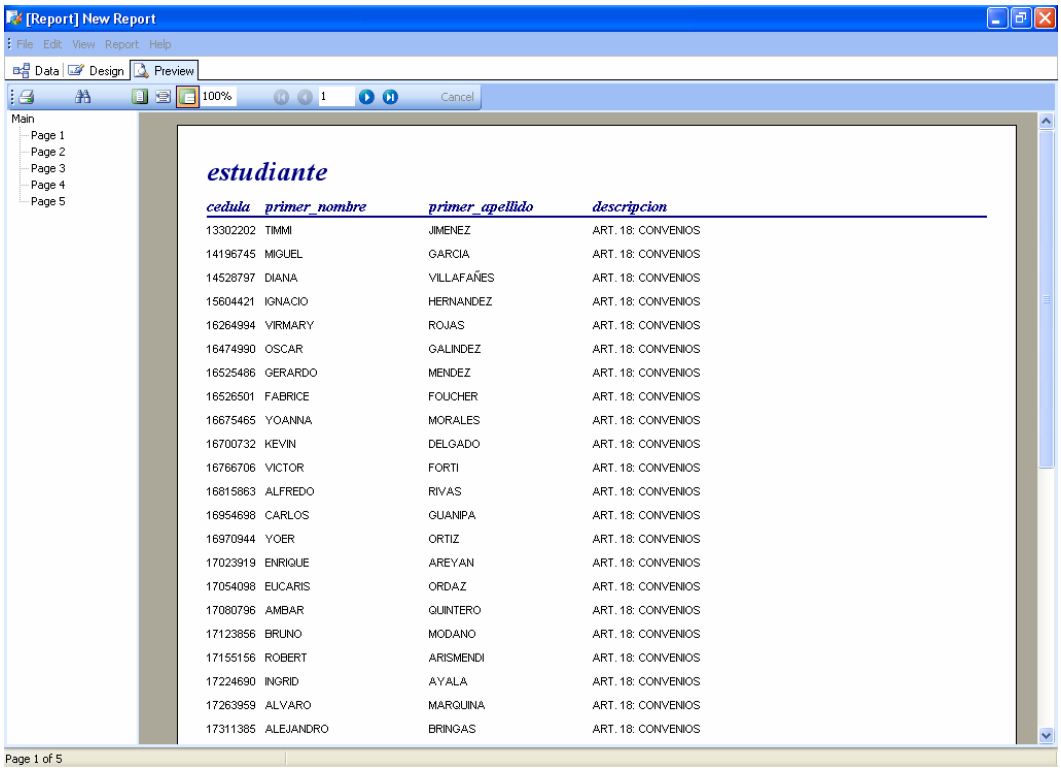


Figura Nº 2-16. Resultado del reporte de listado en Navicat.

En cambio, si se elige crear un reporte de tipo de tabla cruzada, en Navicat el proceso es básicamente el mismo, sólo cambia radicalmente al momento de seleccionar los campos a incluir en el reporte. Hay que ubicar los campos del resultado de la consulta en las filas, columnas o valores calculados según sea el caso, esto se hace mediante la selección y arrastre del campo de una lista de la izquierda sobre las celdas del gráfico (ver Figura Nº 2-17).

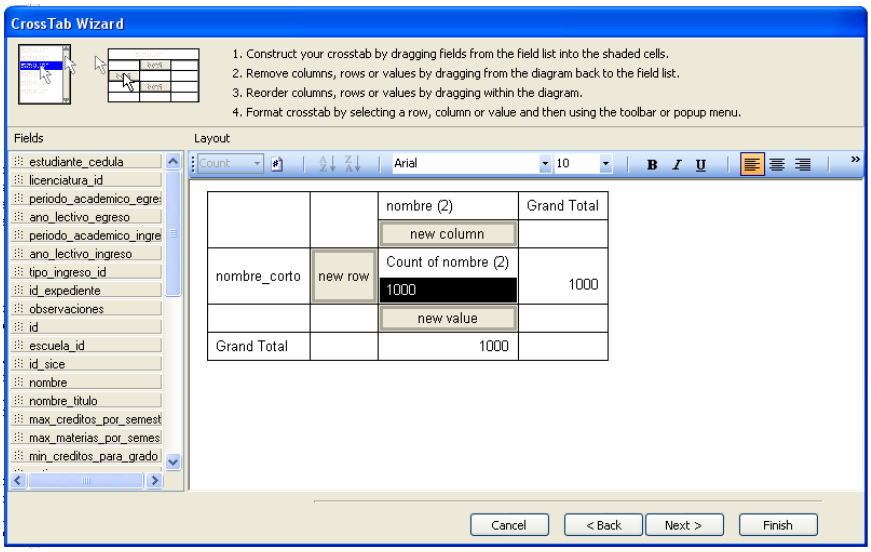
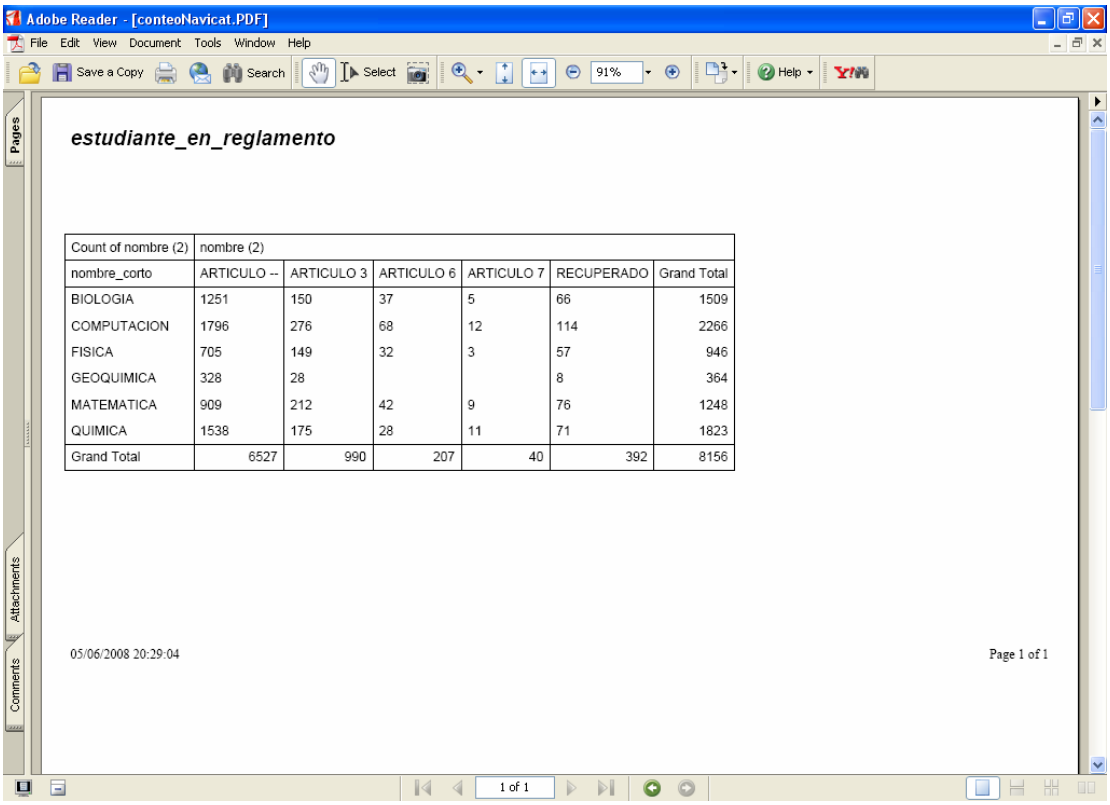


Figura Nº 2-17. Disposición de los campos en la tabla del reporte de tabla cruzada de Navicat.

Una vez elaborado el reporte Navicat permite generar el reporte en varios formatos (ver Tabla Nº 2-2).

Formato	Características
PDF	Se genera el reporte manteniendo tal cual todas las características el diseño elaborado en la aplicación. Permite ver gráficos, imágenes, etc.
HTML	Se genera el reporte manteniendo tal cual todas las características el diseño elaborado en la aplicación. Permite ver gráficos, imágenes, etc.
Archive	El reporte se genera en un formato especial que puede ser visualizado sólo por una aplicación Navicat.

Tabla Nº 2-2. Formatos soportados por Navicat



Count of nombre (2)	nombre (2)					
nombre_corto	ARTICULO --	ARTICULO 3	ARTICULO 6	ARTICULO 7	RECUPERADO	Grand Total
BIOLOGIA	1251	150	37	5	66	1509
COMPUTACION	1796	276	68	12	114	2266
FISICA	705	149	32	3	57	946
GEOQUIMICA	328	28			8	364
MATEMATICA	909	212	42	9	76	1248
QUIMICA	1538	175	28	11	71	1823
Grand Total	6527	990	207	40	392	8156

Figura Nº 2-18. Reporte de tabla cruzada exportado a formato PDF.

2.3.3 **Evaluación de la Herramienta** [Márquez, 2008]

Mediante la elaboración de estos reportes, se pueden definir las siguientes ventajas y desventajas de la herramienta.

- **Ventajas**
  - La herramienta es de fácil uso, aunque la curva de aprendizaje es mayor en comparación con Crystal Reports.
  - Permite la generación de reportes de listados y de conteos mediante secuencias de pasos (wizards) especializados para cada caso.
  - Permite la generación de gráficos a partir de los datos consultados.
  - Permite elaborar reportes a partir bien sea de las tablas o a partir de una consulta escrita en lenguaje de consulta SQL.
  - Es una herramienta que además de manejar reportes permite la administración de la Base de Datos MySQL, como consecuencia la conexión con la base de datos utilizada es directa.

- La herramienta permite al usuario definir y eliminar los enlaces entre las tablas seleccionadas al elaborar un reporte. La herramienta sólo sugiere al usuario los enlaces entre las tablas seleccionadas si los nombres coinciden.

- **Desventajas**
  - Permite obtener los datos sólo desde una Base de Datos MySQL.
  - Poca variedad de formatos en los que se puede generar un reporte, PDF y HTML son los únicos formatos en los que los reportes pueden ser vistos fuera de Navicat.
  - La herramienta permite la edición del formato del reporte, pero no es de forma sencilla. En el caso de los reportes de tipo Tabla Cruzada no se puede eliminar un elemento de formato que muestra el nombre del campo en conjunto con los valores que toma el campo, tanto en las filas como en las columnas.
  - Es una herramienta propietario que tiene un costo alrededor de $139 la versión Navicat 8.0 utilizada.
  - La herramienta tiene separados los procesos de elaboración de la consulta y de elaboración del reporte, lo cual en principio causa confusión y puede ser contraproducente, ya que no se puede generar el reporte de forma continua desde la elaboración de la consulta hasta el diseño del mismo en una secuencia ordenada de pasos.

2.4 SQL Manager for MySQL 2007

2.4.1 Características

a. **Tipo de Licencia:** Propietario.
b. **Empresa desarrolladora:** EMS (Electronic MicroSystems).
c. **Año de publicación:** 2007.
d. **Versión utilizada:** 4.1.2.1.
e. **Características generales:** [EMS, 2008]

SQL Manager for MySQL 2007 es una herramienta que permite el desarrollo y administración de Bases de Datos MySQL disponible para Microsoft Windows y las versiones de MySQL con las que puede trabajar la aplicación van desde MySQL 3.23 a MySQL 6.0. Esta herramienta posee entre sus componentes uno que permite la generación de reportes

Es una herramienta bastante intuitiva que permite crear, navegar sobre bases de datos y ejecutar consultas SQL de una forma sencilla, pero es lo suficientemente sofisticada para cumplir con los requerimientos para el manejo de la base de datos MySQL. Soporta Script SQL, construcciones visuales de consulta SQL, extrae o imprime metadatos, exporta/importa datos, mantenimiento y gestión de privilegios de usuarios, etc.

2.4.2 Proceso de generación de reportes [Márquez, 2008]

Para crear un reporte hay que conectarse a la base de datos y luego seleccionar la opción de Report Designer en el menú de herramientas de la aplicación, para de esta forma tener acceso al componente de reportes.

La herramienta alegaba proveer wizards para llevar a cabo los reportes, pero durante las pruebas, no se logró poder generar ningún reporte mediante estos, por lo que el proceso de generación de un reporte con esta herramienta tuvo que ser elaborado sin ningún tipo de asistencia, como lo haría un usuario experto.

Una vez dentro del componente de reportes llamado “Report Designer”, lo primero que hay que hacer es crear una consulta, para lo cual hay que definir una conexión a la base de datos y colocar los datos para realizar la conexión a ella.

Luego se elabora la consulta, para ello se debe copiar la consulta en lenguaje SQL, el programa no provee ningún mecanismo a este nivel para apoyar el proceso de crear la consulta (ver Figura Nº 2-19).

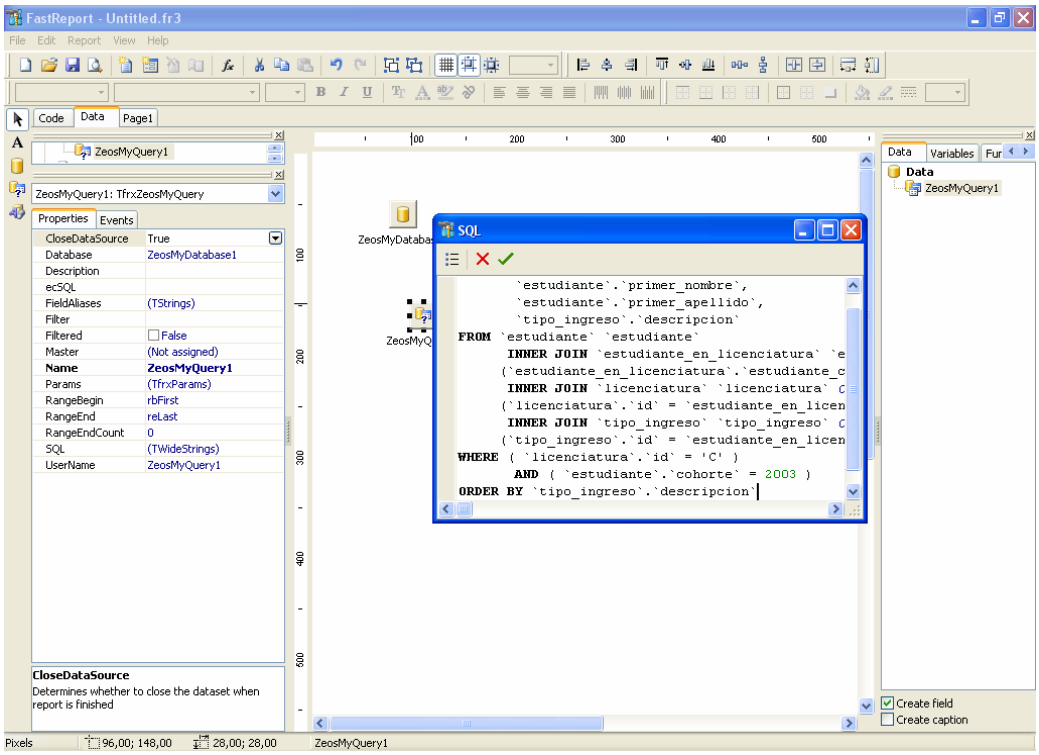


Figura Nº 2-19. Definición de la consulta en SQL Manager for MySQL 2007.

Luego de escribir la consulta, se selecciona la pestaña correspondiente a la edición del reporte (Page 1) y se colocan las partes del reporte que se consideren necesarias, estas partes reciben el nombre de bandas, por ejemplo el pie de página, la cabecera, etc.

Al agregar la banda Master Data, que es la banda donde van los datos, se debe seleccionar de una lista la consulta que se elaboró en pasos anteriores. Los campos que son producto de la consulta, que aparecen en la parte derecha de la herramienta, pueden ser seleccionados y colocados dentro de la banda Master Data (ver Figura Nº 2-20).

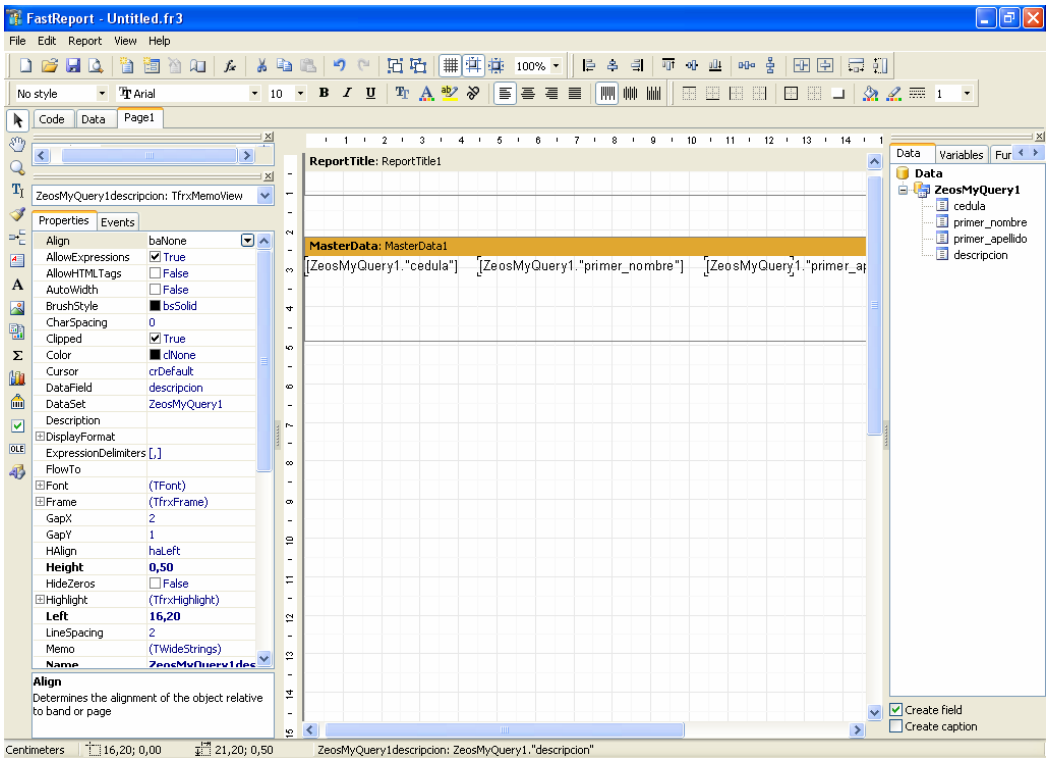


Figura Nº 2-20. Selección de campos en SQL Manager for MySQL 2007.

Al seleccionar la opción de vista preliminar (Preview) se puede observar el reporte generado (ver Figura Nº 2-21).

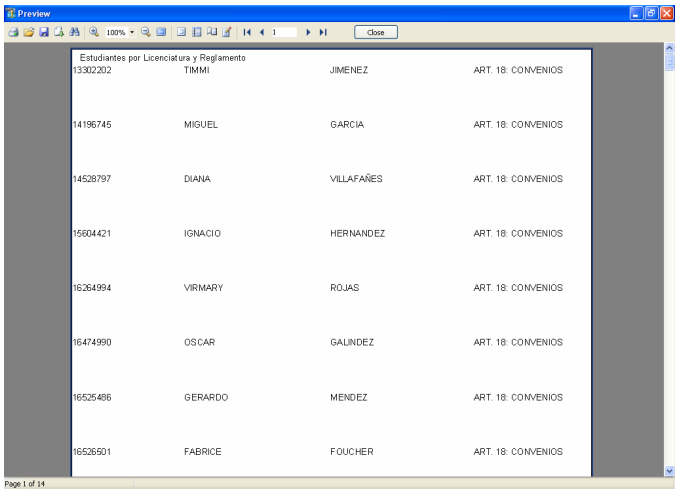


Figura Nº 2-21. Resultado del reporte de listado en SQL Manager for MySQL 2007.

La herramienta sólo provee este tipo de reporte de lista de elementos, no provee mecanismos que permitan generar reportes del tipo de tabla cruzada, como las otras herramientas estudiadas.

El componente para generar reportes permite exportarlo a varios formatos (ver Tabla Nº 2-3).

Formato	Características
PDF	Se genera el reporte manteniendo tal cual todas las características el diseño elaborado en la aplicación. Permite ver gráficos, imágenes, etc.
HTML	Se genera el reporte manteniendo tal cual todas las características el diseño elaborado en la aplicación. Permite ver gráficos, imágenes, etc.
Excel	Los datos se almacenan, siguiendo el formato de Excel, en las celdas manteniendo tal cual todas las características el diseño elaborado en la aplicación.
RTF (Rich Text Format)	Formato en el que el reporte mantiene todas las características de diseño elaboradas al generar el reporte. Este formato es útil si se desea que el reporte pueda ser visualizado mediante procesadores de texto como Word (paquete Office de Microsoft) o Writer (paquete OpenOffice de Sun Microsystems).
CSV (Comma Separated Values)	Este formato sólo almacena los datos que fueron seleccionados al generar el reporte, es decir, almacena campo a campo de los registros producto de la consulta realizada, separados por coma.
BMP image, JPEG image, TIFF image, Gif image	En este formato se emite el reporte como si fuera una imagen.
Texto sin formato (Text file)	Este formato sólo almacena los datos que fueron seleccionados al generar el reporte, es decir, almacena campo a campo de los registros producto de la consulta realizada, en formato de texto sin formato.
Open Document Format (ODF)	Este formato es el diseñado por OASIS que está basado en XML y que es usado en el paquete OpenOffice de Sun Microsystems.

Tabla Nº 2-3. Formatos soportados por SQL Manager for MySQL 2007.

2.4.3 Evaluación de la Herramienta [Márquez, 2008]

Durante la elaboración de estos reportes, se pueden definir las siguientes características de la herramienta.

- **Ventajas**
  - Amplia variedad de formatos en los que se puede generar un reporte.

- **Desventajas**
  - La herramienta es de uso complicado, la curva de aprendizaje es la mayor en comparación con otras herramientas.
  - Permite obtener los datos sólo desde una Base de Datos MySQL.
  - La herramienta permite la edición del formato del reporte, pero no es de una de forma sencilla.
  - Es una herramienta propietario que tiene un costo, alrededor de $129 la versión utilizada.
  - La herramienta tiene el componente de generación de reportes de forma muy separada al resto de la aplicación, de forma que aunque la aplicación esté conectada a la Base de datos dentro del componente de reportes, se hace necesario declarar y establecer nuevamente la conexión a la Base de Datos.
  - Las secuencias de pasos (wizards) que posee la herramienta son confusos y propensos a errores, de forma tal que se optó por generar los reportes de prueba sin utilizarlos.
  - No se pudo elaborar algunos de los reportes de prueba.

2.5 Cuadro comparativo de las herramientas [Márquez, 2008]

Herramientas Características	Crystal Reports XI	Navicat 8	SQL Manager for MySQL 2007
Formatos a los que pueden ser exportados los reportes	Soporta 8 formatos: PDF, HTML, Excel XLS, Excel Data Only, RTF, CSV, TTX, TXT	Sólo soporta 3 formatos: PDF, HTML y Archive (este último sólo puede ser visualizado por Navicat)	Soporta 8 formatos: PDF, HTML, Excel, RTF, CSV, Imagen (BMP, JPEG, Gif), TXT y ODF
Facilidad de uso	Curva de aprendizaje corta.	Curva de aprendizaje mayor que Crystal Reports.	Curva de aprendizaje mayor que las otras herramientas.
Reportes elaborados	– Listado. – Tabla Cruzada.	– Listado. – Tabla Cruzada	– Listado.
Fuentes de datos	Múltiples.	Base de datos MySQL.	Base de datos MySQL.
Permite generar gráficos a partir de los datos	Si.	Si.	No.
Edición del formato del reporte	Si, de forma sencilla e intuitiva.	Si, de forma un poco más compleja.	Si, de forma un poco más compleja.
Posee secuencias de pasos (wizards) para la elaboración de reportes	Si, son de fácil uso.	Si, son de fácil uso.	Si, son complejos de utilizar.
Permite elaboración de la consulta y diseño del reporte	Si, en una sola secuencia de pasos.	Si, en dos secuencias por separado.	Si, en dos secuencias por separado.

Tabla Nº 2-4. Comparación de las herramientas.

2.6 El patrón de diseño: Wizard [Tidwell, 2005]

Las herramientas generadoras de reporte que fueron estudiadas, toman en cuenta que deben apoyar al usuario en la creación de un reporte y han desarrollado para ello secuencias de pasos ordenados que sirven de guía al usuario, organizando el proceso, aligerando la carga y la dificultad técnica, lo cual termina facilitando la generación de los reportes. Siendo esta una característica común y que se considera relevante en este tipo de herramientas, se hace un estudio a fondo de esta característica a continuación.

2.6.1 Los wizards y su uso

Los wizards o también llamados asistentes, son programas que conducen a un usuario a través de una serie de acciones con el propósito de completar una tarea compleja. Por ejemplo, en el sistema operativo Windows se utilizan para conectarse a recursos de red, al momento de instalar nuevos programas, al agregar una impresora (ver Figura Nº 2-22), etc.

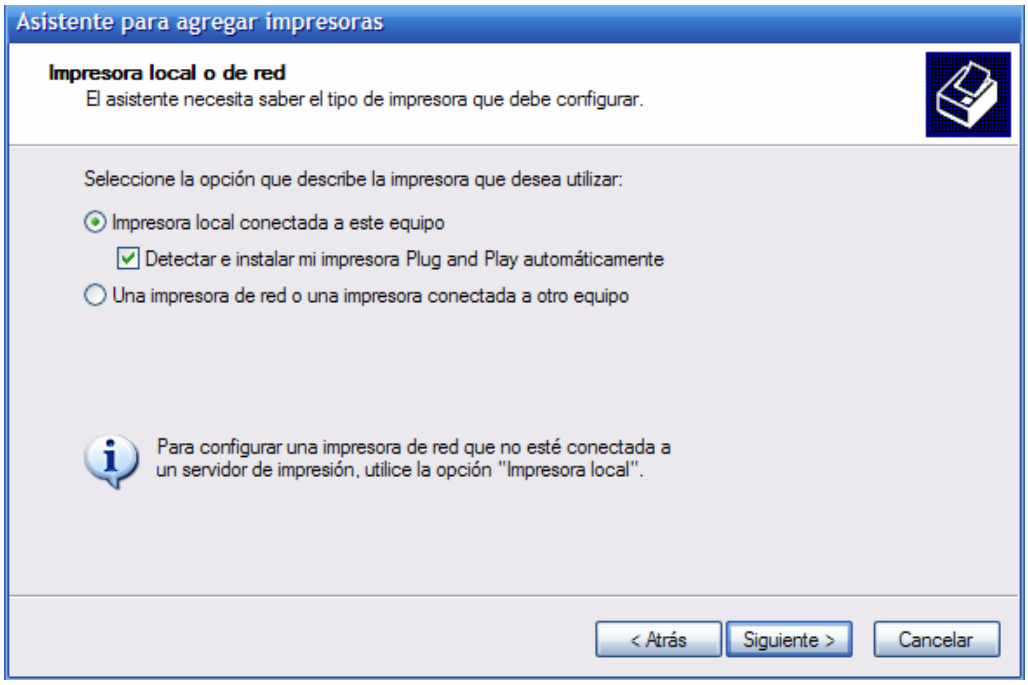


Figura Nº 2-22. Ejemplo de uso del patrón en el sistema operativo Windows.

Este patrón aplica cuando se diseña una interfaz de usuario para una tarea que es larga o complicada, en la que el usuario deberá tomar varias decisiones y realizar varias sub-tareas, y en la que el usuario no tiene un conocimiento avanzado, generalmente son tareas que no se hacen de forma muy común o sobre la que no se desea que el usuario tenga un control muy detallado.

Las tareas que parecen estar bien adaptadas para este patrón tienden a estar divididas en múltiples caminos o a ser largas y tediosas. El objetivo fundamental de un wizard es guiar al usuario a través de la interfaz paso a paso, haciendo tareas en un orden previamente establecido.

Este busca simplificar la tarea al usuario aplicando el principio de “Divide y Vencerás”, ya que, a través de la división de la tarea en una secuencia de pequeñas tareas, cada una de las cuales puede ser resuelta por el usuario en un relativo pequeño espacio de tiempo, se simplifica la tarea a lograr. Todo lo que el usuario debe hacer es realizar las tareas de cada paso, confiando de que si sigue las instrucciones todo resultará de manera correcta.

2.6.2 Consideraciones de diseño de los wizards

La principal característica en este tipo de interfaces es que el usuario debe estar dispuesto a entregar el control sobre qué ocurre y cuándo. En muchos contextos, esto funciona, en especial cuando un usuario desea ser guiado y desea saber qué hacer sin tener que pensar mucho. Pero en otros contextos, puede ser contraproducente. Comúnmente los usuarios expertos encuentran los wizards bastante rígidos y limitantes.

La parte más complicada del diseño de este tipo de interfaz de usuario es establecer un balance en la cantidad de las tareas a realizar en cada paso y la cantidad de pasos. No tiene mucho sentido realizar un wizard de sólo dos pasos, pero a su vez un wizard de quince pasos es tedioso, y además hay que tener en cuenta que un paso no puede tener demasiadas tareas, ya que, se pierden los beneficios del uso de este patrón.

Al momento de diseñar un wizard también hay que tener en cuenta que al momento de que el usuario deba introducir algún valor, se analicen cuales son las posibles opciones y seleccionar por defecto la opción más probable para aligerar la carga y agilizar el proceso.

Hay que recordar que cuando se utiliza un wizard generalmente los usuarios no son expertos y los mensajes deben utilizar un lenguaje claro y sencillo que pueda ser comprendido por el usuario del mismo. El uso de buenas metáforas e iconos pueden ayudar a mejorar y aligerar las tareas.

Los wizards generalmente presentan cada paso en una página separada, enlazadas mediante el uso de botones de Atrás y de Siguiente (ver Figura Nº 2-23).

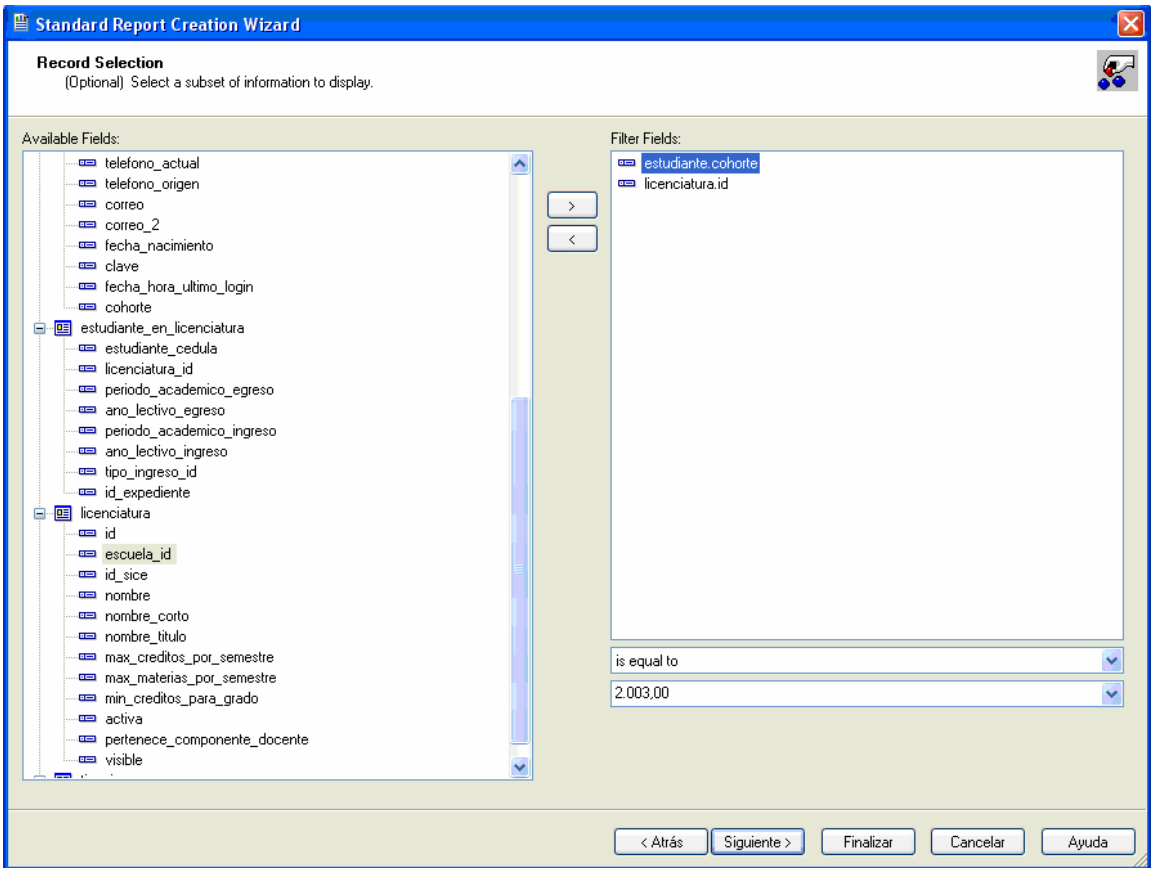


Figura Nº 2-23. Uso del patrón Wizard en la interfaz de Crystal Reports.

En los wizards se tiene que permitir al usuario que se mueva hacia atrás y hacia adelante, durante toda la secuencia de la tarea. No hay nada más frustrante que tener que comenzar una tarea nuevamente simplemente porque la aplicación no permite al usuario que cambie de opinión acerca de una decisión previa. Los botones de regresar o atrás son un elemento que debe aparecer en todo wizard de este tipo. Adicionalmente y de forma opcional, se puede tener una indicación sobre qué pasos faltan o qué pasos han sido superados.

CAPÍTULO 3

IDENTIFICACIÓN DE LOS REPORTES

En este capítulo se describe cómo se realiza actualmente el proceso de generación de reportes y se analizan los inconvenientes asociados a dicho proceso dentro de la División de Control de Estudios de la Facultad de Ciencias. Adicionalmente se presentan diversas clasificaciones de los reportes que se generan

3.1 Análisis de la situación actual

La División de Control de Estudios de la Facultad de Ciencias, dispone de una aplicación Web denominada CONEST que automatiza los procesos de gestión académica de los estudiantes y docentes de las seis licenciaturas que conforman el pregrado de esta facultad (Biología, Computación, Física, Geoquímica, Matemática y Química), también maneja toda la información correspondiente a los estudiantes, historiales académicos, así como información de las materias, los docentes, las aulas, etc. Debido a esto, los docentes y los distintos departamentos de las Escuelas y de la Facultad, solicitan a la División de Control de Estudios de forma frecuente una serie de reportes, relacionados en su mayor medida con los datos académicos de los estudiantes.

Aunado a esto, internamente para los distintos procesos que lleva a cabo la División de Control de Estudios, también es necesario emitir una serie de reportes, que incluyen datos de los estudiantes, de las materias y reportes internos relacionados directamente con el sistema como tal.

En la División de Control de Estudios al recibir las solicitudes de reportes se lleva a cabo un proceso de análisis de los datos solicitados y se delega sobre varios integrantes la responsabilidad de emitir los reportes de acuerdo a su complejidad.

Es importante resaltar que los datos que se consultan se encuentran almacenados en un repositorio de datos relacional y cuyo modelo lógico de datos es de un tamaño considerablemente grande, esto debido en parte a que la base de datos se encuentra normalizada y en consecuencia los datos se encuentran almacenados en un gran número de tablas pequeñas, que además de ser simples y estables, son fáciles de mantener. Para el momento de realizar esta investigación el modelo de datos tenía aproximadamente 140 tablas, lo cual añade un grado de dificultad mayor y exigen conocimientos avanzados en la elaboración de consultas a Bases de Datos al momento de realizar los reportes.

Tomando esto en cuenta, las personas encargadas de esta tarea de elaborar reportes utilizan aplicaciones especializadas que permiten hacer consultas en el lenguaje de consultas nativo de las bases de datos relacionales (SQL) e interactúan directamente con la Base de Datos lo cual representa un riesgo para la integridad de los datos.

Una vez que se tiene la consulta elaborada en lenguaje SQL, esta es ejecutada y los resultados son copiados en aplicaciones de oficina en los cuales se les da formato y en algunos casos se generan gráficos. Este último paso es propenso a que se cometan errores y además se invierte mucho tiempo dándole formato a los datos.

En la Figura Nº 3-1 se representa gráficamente el proceso actual de generación de reportes.

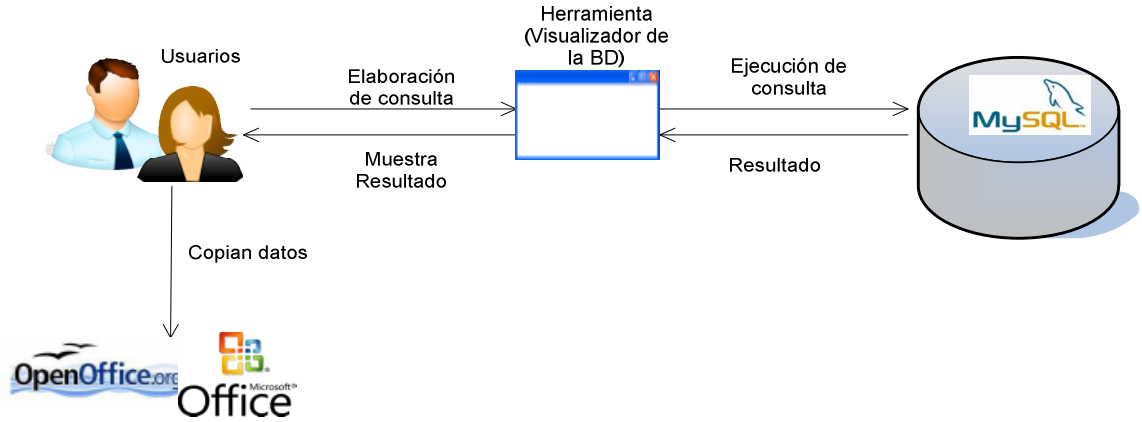


Figura Nº 3-1. Proceso actual de generación de reportes.

3.2 Entidades [Márquez, 2008]

La mayoría de los reportes que se emiten en el ámbito de los datos académicos, según las personas encargadas de elaborarlos, tienen como resultado datos de los estudiantes, las materias y los docentes o involucran relaciones entre estos elementos. En base a esto se identificaron las siguientes entidades fundamentales:

- Estudiante (Ver Figura 3-2).
- Materia (Ver Figura 3-3).
- Docente (Ver Figura 3-4).

Cada una de estas entidades tiene un conjunto de atributos, los cuales son objeto de restricciones al momento de emitir los reportes. Se identifican dos tipos de atributos:

- **Restringibles:** Son aquellos atributos sobre los cuales se puede definir una regla lógica la cual permite seleccionar los elementos que la cumplan o no. Estos atributos pueden ser producto de un cálculo, como por ejemplo la eficiencia, el promedio general de notas, etc. , o pueden ser valores simples como el nombre, el apellido, etc. Los reportes generalmente tienen definida una restricción de tiempo, con la finalidad de poder analizar los datos de alguna entidad en un período académico y año lectivo, o un conjunto de ellos, debido al contexto universitario de la Facultad de Ciencias. Esto es necesario porque tras cada período académico las entidades sufren cambios de estado relevantes que son almacenados y que posteriormente se requieren recuperar para su análisis.

- **No Restringibles:** Son aquellos atributos sobre los cuales no se puede definir una regla lógica que permita la selección de elementos.

Sobre esta clasificación se detallarán los atributos de las entidades anteriormente identificadas en las figuras que se muestran a continuación.

3.2.1 Entidad Estudiante

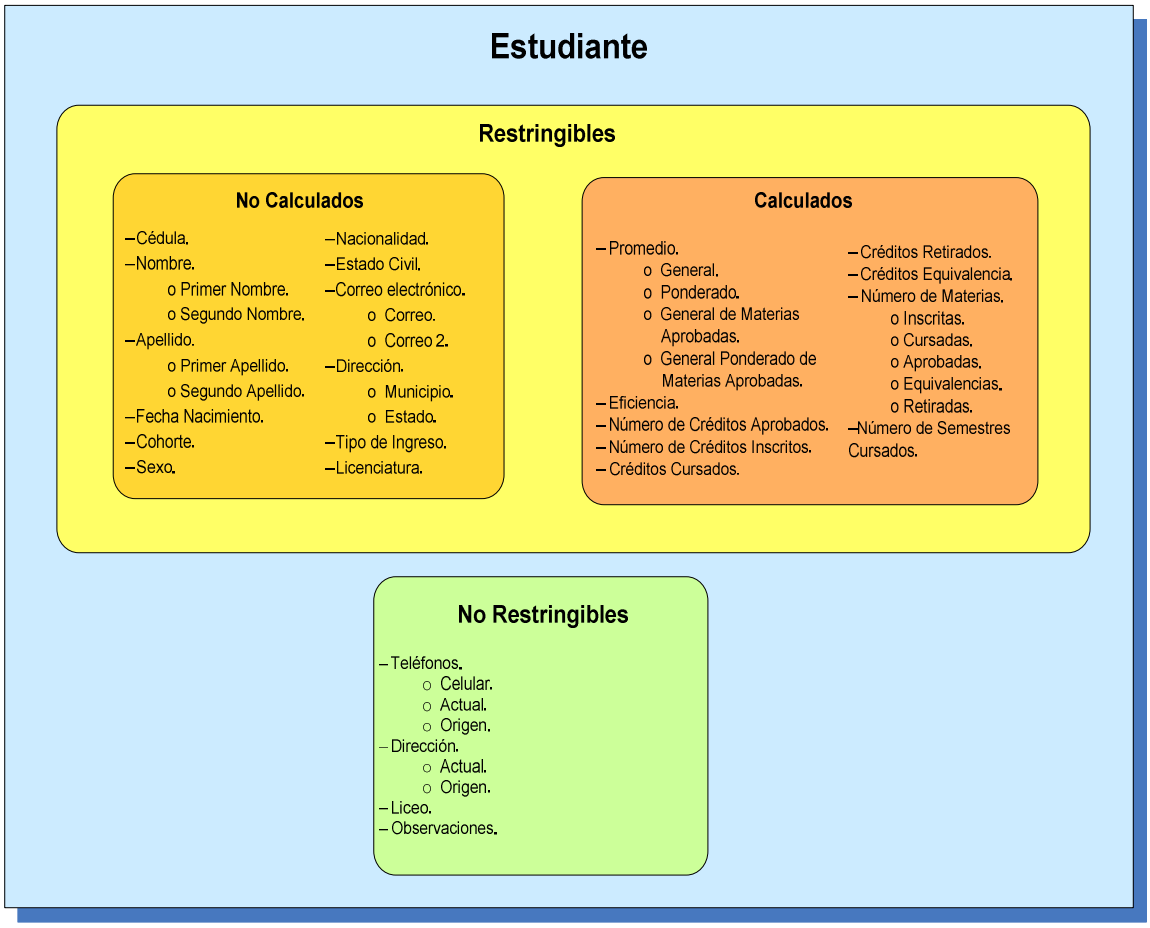


Figura Nº 3-2. Entidad Estudiante.

3.2.2 Entidad Materia

Figura Nº 3-3. Entidad Materia.

3.2.3 Entidad Docente

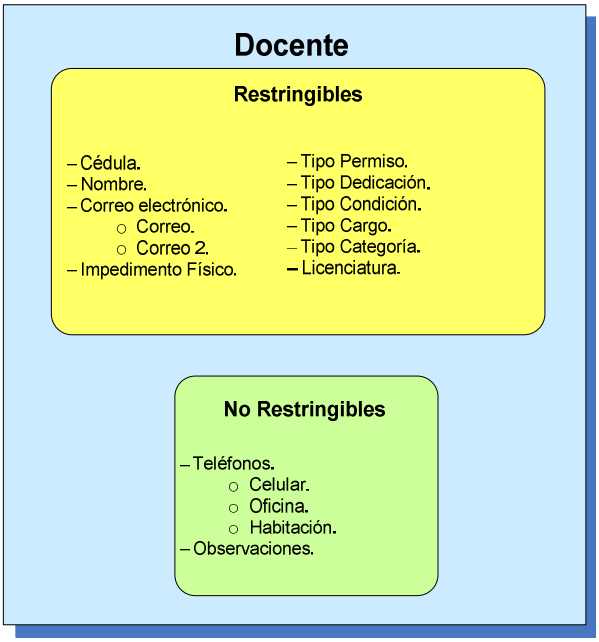


Figura Nº 3-4. Entidad Docente.

Es importante resaltar que existen atributos que poseen las entidades que son resultado de la relación con las otras entidades, por ejemplo, la entidad estudiante posee atributos calculados como lo son el promedio de notas, cantidad de materias cursadas, etc., que son producto de la relación con otras entidades, en este caso con la entidad materia.

3.3 Clasificaciones de los Reportes [Márquez, 2008]

Los reportes que se emiten actualmente pueden clasificarse de acuerdo a muchos factores y depende del analista y del punto de vista en que se estudie el proceso, las distintas clasificaciones que se pueden tener. En esta investigación se pueden identificar los siguientes tipos de reportes en base a la frecuencia con que son emitidos:

a. Reportes Regulares.
b. Reportes por Solicitud.

3.3.1 Reportes Regulares

Son reportes que se elaboran de forma periódica, es decir, son aquellos que se emiten de forma constante cada cierto tiempo. Por ejemplo: Listado de materias (código, nombre corto y créditos) ofertadas por la Licenciatura de Matemática en un período y año lectivo dado.

3.3.2 Reportes por Solicitud

Son reportes que se elaboran en base a las solicitudes que recibe la División de Control de Estudios. Estos reportes generalmente necesitan que un integrante del grupo de trabajo se dedique a la elaboración de la consulta en el lenguaje SQL y generalmente se emite una sola vez para resolver un requerimiento específico. Por ejemplo: Se solicitan los alumnos que no deben realizar Servicio Comunitario. Otro ejemplo podría ser un listado de los estudiantes que habitan en el Estado Vargas.

Dentro de la clasificación anteriormente expuesta se puede hacer una nueva clasificación en base a los datos que se solicitan en los reportes:

a. Listados.
b. Tabla Cruzada.

3.3.3 Listados

Son reportes en los que el resultado es una lista de elementos, pertenecientes a una entidad, con datos asociados y que pueden o no incluir campos producto de conteos u operaciones aritméticas.

En este tipo de reportes los resultados generalmente son producto de un proceso de selección establecido en base a restricciones. Por ejemplo: Listado de los estudiantes (Apellido, Nombre y Cédula) que están cursando en el período y año actual la materia cuyo nombre_corto es Redes (Ver Figura 3-5).

estudiante

cedula	*primer_nombre*	*primer_apellido*
13302202	TIMMI	JIMENEZ
14196745	MIGUEL	GARCIA
14528797	DIANA	VILLAFAÑES
15604421	IGNACIO	HERNANDEZ
16264994	VIRMARY	ROJAS
16474990	OSCAR	GALINDEZ
16525486	GERARDO	MENDEZ
16526501	FABRICE	FOUCHER
16675465	YOANNA	MORALES
16700732	KEVIN	DELGADO
16766706	VICTOR	FORTI
16815863	ALFREDO	RIVAS
16954698	CARLOS	GUANIPA

Figura Nº 3-5. Ejemplo de Reporte de Listado.

3.3.4 Tabla Cruzada

Son reportes en los que el resultado es producto de la realización de una operación aritmética, generalmente conteo, y que se presenta en forma de tabla. Por ejemplo: Mostrar por cada licenciatura la cantidad de estudiantes según su estado en el sistema, si están en artículo 3, artículo 6, artículo 7, etc., en un año y período académico (Ver Figura 3-6).

	ARTICULO --	ARTICULO 3	ARTICULO 6	ARTICULO 7	RECUPERAD	Total
Total	6.527	990	207	40	392	8.156
BIOLOGIA	1.251	150	37	5	66	1.509
COMPUTACION	1.796	276	68	12	114	2.266
FISICA	705	149	32	3	57	946
GEOQUIMICA	328	28	0	0	8	364
MATEMATICA	909	212	42	9	76	1.248
QUIMICA	1.538	175	28	11	71	1.823

Figura Nº 3-6. Ejemplo de Reporte de Tabla Cruzada.

Para cada una de las entidades anteriormente explicadas se identificaron una serie de reportes, que se enmarcan dentro de las clasificaciones expuestas y se muestran a continuación.

Entidad Estudiante

Listados:

- Listado de estudiantes por Licenciatura: Dada una, varias o todas las Licenciaturas, listar los estudiantes inscritos en un año lectivo y período académico.

- Listado de estudiantes por Materia: Dada una o varias Materias, listar los estudiantes inscritos en la misma, es decir, que estén viendo todas las materias dadas o al menos una de ellas.

  Entre los atributos académicos del estudiante que pueden ser consultados en este reporte se encuentran:

  - Calificación del Estudiante en la materia.
    - Definitiva.
    - Final.
    - Reparación.
    - Suficiencia.
    - Equivalencia.

  - Sección en la que cursó una materia.

  - En este reporte también se podría visualizar el estado del Estudiante:
    - Retirado.
    - Aprobado.
    - Aplazado.
    - Sin Calificar.

- Listado de estudiantes por Tipo de Ingreso.
  - CNU.
  - Convenios.
  - Prueba Interna.
  - Samuel Robinson.

- Listado de estudiantes incursos en Reglamento de Permanencia.
    - Artículo 3.
    - Artículo 6.
    - Artículo 7.

- Listado de estudiantes en Grado

  Dado a que a los estudiantes se les puede otorgar o no un premio se tiene un reporte por tipo de premio académico obtenido.

    - Suma Cum Laude.
    - Magna Cum Laude.
    - Alto Rendimiento Académico.
    - Premio Especial de Graduación.

Tablas cruzadas y cálculos:

- Cantidad de Estudiantes
    - Por tipo de Ingreso.
    - Incursos en Reglamento de Permanencia.

- Promedios de notas por Licenciatura para graduandos.
- Promedios de Número de Semestres Cursados por Licenciatura para graduandos.

Entidad Materia:

Listados:

- Listados de Materias por Licenciatura Origen: Dada una, varias o todas las Licenciaturas, listar todas las materias ofertadas en un año lectivo y período académico.

- Listados de Materias que se dictan en cada Licenciatura (Pensum): Dada una, varias o todas las Licenciaturas, listar todas las materias.

- Listados de Materias dictadas en un año lectivo y período académico ordenadas por tipo (Obligatoria, Electiva, Complementaria, PCI, etc.)

- Listados de Materias por Docente: Dado un Docente, listar todas las materias que son dictadas por él en un año lectivo y período académico.

Tablas cruzadas y cálculos:

- Cantidad de Estudiantes
  - Aprobados.
  - Aplazados.
  - Retiros.
  - Reparaciones.
  - Final.
  - Equivalencia.

- Cantidad de Secciones.

Entidad Docente:

Listados:

- Listados de Docentes por Licenciatura: Dada una, varias o todas las Licenciaturas, listar todos los docentes que dictaron al menos una materia en un año lectivo y período académico.

- Listados de Docentes por Materia: Dada una o varias materias, listar los docentes que la dictaron en un año lectivo y período académico.

Para cada uno de los reportes que se listaron anteriormente, se pueden definir restricciones sobre los atributos de las entidades involucradas y a su vez cada uno de los reportes pueden ser combinados (según el caso) con otro para producir un resultado más especifico, por citar un ejemplo se pueden combinar el listado de estudiantes por materia y definir una restricción sobre el tipo de materia lo cual da como resultado un reporte totalmente distinto, por tanto la lista anterior pretende ser sólo una guía general de los reportes a tener en cuenta en el sistema.

3.4 El Sistema Manejador de Base de Datos MySQL [MySQL, 2008]

MySQL es el sistema manejador de bases de datos relacional de software libre más popular; lo desarrolla, distribuye y soporta MySQL AB. Esta es una compañía comercial, fundada por los desarrolladores de MySQL.

Una base de datos es una colección estructurada de datos. Puede ser cualquier cosa, desde una simple lista de compra a una galería de pintura o las más vastas cantidades de información en una red corporativa. Para añadir, acceder, y procesar los datos almacenados en una base de datos, se necesita un sistema de gestión de base de datos como MySQL Server. Al ser los computadores muy buenos en tratar grandes cantidades de datos, los sistemas de gestión de bases de datos juegan un papel central en computación, como aplicaciones autónomas o como parte de otras aplicaciones.

MySQL es un sistema de gestión de bases de datos relacional. Una base de datos relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén. Esto añade velocidad y flexibilidad.

La parte SQL de "MySQL" se refiere a "Structured Query Language". SQL es el lenguaje estandarizado más común para acceder a bases de datos relacionales y está definido por el estándar ANSI/ISO SQL.

Entre sus principales características se pueden mencionar:

- El servidor de base de datos MySQL es rápido, fiable y fácil de usar.
- Funciona en diferentes plataformas, Windows, Linux, Unix, Solaris.
- Está escrito en el Lenguaje C y en C++.
- Trabaja en entornos cliente/servidor o incrustados.
- El software de bases de datos MySQL es un sistema cliente/servidor que consiste en un servidor SQL que soporta múltiples hilos de ejecución que trabaja con diferentes programas y bibliotecas cliente, herramientas administrativas y un amplio abanico de interfaces de programación para aplicaciones (APIs). Existen APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl.
- Una gran cantidad de software de contribuciones está disponible para MySQL por lo que una gran cantidad de aplicaciones o lenguajes de programación soportan el manejo de datos con el servidor de base de datos MySQL.
- Proporciona sistemas de almacenamiento transaccionales y no transaccionales.

- Maneja registros de longitud fija y longitud variable, con diversos tipos de datos en las columnas.
- Soporte completo para distintos conjuntos de caracteres, incluyendo latin1 (ISO-8859-1), german, big5, ujis, y más.
- Posee un sistema de privilegios y contraseñas que es muy flexible y seguro. Las contraseñas son seguras porque todo el tráfico de contraseñas está encriptado cuando se conecta con un servidor.
- Provee varias opciones de conectividad. Los clientes se pueden conectar con el servidor MySQL usando sockets TCP/IP en cualquier plataforma o también se puede utilizar el conector ODBC (MyODBC), el cual proporciona a MySQL soporte para programas clientes que usen conexiones ODBC (Open Database Connectivity), por ejemplo para conectar Crystal Reports IX al servidor MySQL.

3.4.1 El manejo de Vistas [MySQL, 2008]

Una característica del manejador de Base de Datos relacional de MySQL es el manejo de vistas, esta característica juega un papel importante en el desarrollo del presente Trabajo Especial de Grado. Estas vistas fueron introducidas en la versión 5.0 del servidor de base de datos MySQL.

Una vista es un objeto de la base de datos que se define mediante una sentencia del tipo SELECT que agrupa o selecciona un conjunto de datos.

La sintaxis para crear una vista es la siguiente:

```
CREATE
[ALGORITHM = {MERGE  | TEMPTABLE | UNDEFINED}]
VIEW [nombre_base_de_datos].[nombre_de_la_vista]
AS
[SELECT  sentencia]
```

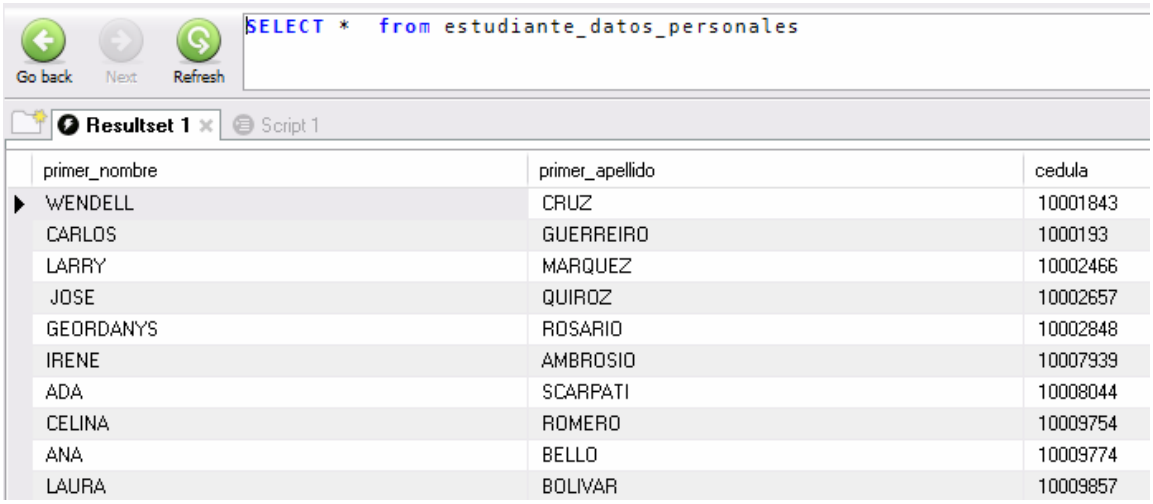
Por ejemplo, podemos crear una vista de la siguiente forma:

```
CREATE
ALGORITHM = UNDEFINED
VIEW conest.estudiante_datos_personales
AS
```

SELECT `estudiante`.`primer_nombre`, `estudiante`.`primer_apellido`,`estudiante`.`cedula` from estudiante

Las tablas y las vistas comparten el mismo espacio de nombres en la base de datos, por eso, una base de datos no puede contener una tabla y una vista con el mismo nombre.

Al igual que las tablas, las vistas no pueden tener nombres de columnas duplicados. Por defecto, los nombres de las columnas devueltos por la sentencia SELECT se usan para las columnas de la vista, aunque pueden renombrarse (Ver Figura 3-7).



primer_nombre	primer_apellido	cedula
WENDELL	CRUZ	10001843
CARLOS	GUERREIRO	1000193
LARRY	MARQUEZ	10002466
JOSE	QUIROZ	10002657
GEORDANYS	ROSARIO	10002848
IRENE	AMBROSIO	10007939
ADA	SCARPATI	10008044
CELINA	ROMERO	10009754
ANA	BELLO	10009774
LAURA	BOLIVAR	10009857

Figura Nº 3-7. Resultado de consultar la vista creada en el ejemplo.

Las columnas devueltas por la sentencia SELECT pueden ser simples referencias a columnas de la tabla, pero también pueden ser expresiones conteniendo funciones, constantes, operadores, etc.

Las vistas pueden crearse a partir de varios tipos de sentencias SELECT. Pueden hacer referencia a tablas o a otras vistas. Pueden usar combinaciones y subconsultas. La sentencia SELECT inclusive no necesita hacer referencia a otras tablas, por ejemplo se puede definir una vista que selecciona dos columnas de otra tabla, así como una expresión calculada a partir de ellas.

Una vista puede hacer referencia a tablas o vistas en otras bases de datos precediendo el nombre de la tabla o vista con el nombre de la base de datos apropiada.

La definición de una vista está sujeta a las siguientes limitaciones:

- La sentencia SELECT no puede contener una subconsulta en su cláusula FROM.
- La sentencia SELECT no puede hacer referencia a variables del sistema o del usuario.
- La sentencia SELECT no puede hacer referencia a parámetros de sentencia preparados.
- Dentro de una rutina almacenada, la definición no puede hacer referencia a parámetros de la rutina o a variables locales.
- Cualquier tabla o vista referenciada por la definición debe existir.
- La definición no puede hacer referencia a una tabla temporal, y tampoco se puede crear una vista temporal.
- Las tablas mencionadas en la definición de la vista deben existir siempre.
- No se puede asociar un disparador (trigger) con una vista.

Existen tres tipos de vistas según el valor de la cláusula opcional ALGORITHM, la cual es una extensión de MySQL al SQL estándar. Dicho valor es muy importante, ya que afecta la manera en que MySQL procesa la vista y se explica cada uno a continuación:

- **MERGE:** el texto de una sentencia que haga referencia a la vista y la definición de la vista son mezclados de forma que parte de la definición de la vista reemplaza las partes correspondientes de la consulta.

- **TEMPTABLE**: los resultados devueltos por la vista son colocados en una tabla temporal, la cual es luego utilizada para ejecutar la sentencia.

- **UNDEFINED:** es el valor por defecto si no se encuentra presente la cláusula ALGORITHM e indica que MySQL determinará el algoritmo que utilizará. En ese caso, MySQL prefiere MERGE por sobre TEMPTABLE si es posible, ya que MERGE por lo general es más eficiente y porque la vista no puede ser actualizable si se emplea una tabla temporal.

Una razón para elegir explícitamente TEMPTABLE es que los bloqueos en tablas subyacentes pueden ser liberados después que la tabla temporal fue creada, y antes de que sea usada para terminar el procesamiento de la sentencia. Esto podría resultar en una liberación del bloqueo más rápida que en el algoritmo MERGE, de modo que otros clientes que utilicen la vista no estarán bloqueados mucho tiempo.

El algoritmo de una vista puede ser UNDEFINED en tres situaciones:

- No se encuentra presente una cláusula ALGORITHM en la sentencia CREATE VIEW.
- La sentencia CREATE VIEW tiene explícitamente una cláusula ALGORITHM = UNDEFINED.

- Se especificó ALGORITHM = MERGE para una vista que solamente puede ser procesada usando una tabla temporal. En este caso, MySQL emite una advertencia y establece el algoritmo en UNDEFINED.

Como se dijo anteriormente, el algoritmo MERGE provoca que las partes correspondientes de la definición de la vista se combinen dentro de la sentencia que hace referencia a la vista. El siguiente ejemplo muestra brevemente cómo funciona el algoritmo MERGE. El ejemplo asume que hay una vista v_merge con esta definición:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Por ejemplo: Suponiendo que se utilice esta sentencia:

```
SELECT * FROM v_merge;
```

MySQL la gestiona del siguiente modo:

- Se convierte v_merge en t.
- Se convierte el * en vc1, vc2, que corresponden a c1, c2.
- Se agrega la cláusula WHERE de la vista.

La sentencia ejecutada resulta ser:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

El algoritmo MERGE necesita una relación uno-a-uno entre los registros de la vista y los registros de la tabla subyacente. Si esta relación no se sostiene, debe emplear una tabla temporal en su lugar. No se tendrá una relación uno-a-uno si la vista contiene cualquiera de estos elementos:

- Funciones agregadas (SUM(), MIN(), MAX(), COUNT(), etcétera).
- DISTINCT
- GROUP BY
- HAVING
- UNION o UNION ALL
- Hace referencia solamente a valores literales (en tal caso, no hay una tabla subyacente)

Los principales problemas de las vistas en MySQL son:

- El procesamiento de vistas no está optimizado:
- No es posible crear un índice en una vista.
- Los índices pueden utilizarse para procesar vistas usando un algoritmo de combinación (MERGE). Sin embargo, una vista que se procesa con el algoritmo de tablas temporales (TEMPTABLE) no es capaz de tomar ventaja de los índices que hacen referencia a las tablas que contiene (aunque los índices pueden ser usados durante la generación de las tablas temporales).

Todo este conjunto de características que poseen las vistas en el manejador MySQL, proveen la posibilidad de que sean utilizadas para el desarrollo de un modelo lógico transparente para el usuario, que permita reflejar las entidades anteriormente identificadas y de esta manera simplificar el proceso de generación de las consultas y obtención de los datos necesarios para la generación de los reportes en la aplicación que se desarrollará en el presente Trabajo Especial de Grado.

Existen otras características adicionales de las vistas pero que escapan del dominio del presente Trabajo Especial de Grado.

PARTE III

MARCO APLICATIVO

CAPÍTULO 4

ADAPTACIÓN XP

En el presente capítulo se describirán los aspectos más relevantes del proceso de desarrollo Programación Extrema (XP) y se plantea una adaptación del proceso para el desarrollo del módulo de generación de reportes.

4.1 Adaptación del Proceso de Desarrollo XP

4.1.1 Programación Extrema [Márquez, 2008]

La Programación Extrema (XP) es una metodología ágil de desarrollo de software que se basa en la simplicidad, la comunicación directa entre las personas involucradas en el proceso y la reutilización del código desarrollado. No se enfoca en la documentación sino en los requerimientos comunicados por el cliente.

El objetivo principal que se persigue es la satisfacción del cliente, por eso tiene mucha importancia la comunicación con los usuarios o clientes. Esta comunicación se va a soportar principalmente en las historias de usuario (del término en inglés User Stories) cuando proviene desde el cliente, y de las entregas y versiones parciales del sistema cuando la comunicación es hacia el cliente.

Este proceso fue diseñado para entregar al cliente el software que necesita y cuando lo necesita, respondiendo muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final del ciclo de la programación. Por otro lado trata de potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores son parte del equipo y están involucrados en el desarrollo del software.

4.1.2 Iteración Programación Extrema

El desarrollo esta basado o compuesto principalmente por iteraciones. La idea de las iteraciones es ir trabajando sobre versiones pequeñas o parciales del sistema hasta llegar al producto final. Durante el desarrollo de la aplicación se reciben los requerimientos y la retroalimentación progresivamente.

Las iteraciones pueden ser de dos tipos principalmente: por objetivos o por lapsos de tiempo. En el desarrollo de esta aplicación las iteraciones estarán basadas en lapsos de tiempo estimados de dos semanas. Durante el tiempo fijado para cada iteración se realizan las implementaciones indicadas en las historias de usuario que abarque.

4.1.3 Historias de Usuarios

Las historias de usuario son la técnica utilizada para especificar los requisitos y funcionalidades que debe cumplir el software, éstas tienen el propósito de describir los requerimientos de los clientes. El contenido de las historias de usuario proviene de los clientes, tal y como ven ellos las necesidades del sistema, por lo tanto son descripciones cortas y escritas en el lenguaje del usuario, sin terminología técnica.

En cada iteración se lleva a cabo un grupo de historias, es decir, en cada iteración se trabaja sobre uno o más requerimientos, tal como se puede apreciar en la tabla 4-1.

Iteración 0	Historia de usuario 1 Historia de usuario 2
Iteración 1	Historia de usuario 3 Historia de usuario 4 …

Tabla Nº 4-1. Iteraciones e historias de usuario.

El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden ser desechadas, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario debe ser lo suficientemente comprensible y estar delimitada para ser desarrollada en un corto período de tiempo.

Cabe destacar que se tienen tres variantes de historia de usuario para los requerimientos:

- Nueva.
- Corrección.
- Mejora.

Con respecto a la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no un único formato a seguir.

En esta adaptación del proceso de desarrollo XP se utilizará el formato de plantilla mostrado en la tabla 4-2.

Id	Fecha	Requerimiento	Tipo

Tabla Nº 4-2. Plantilla para la representación de las historias de usuario.

4.1.4 Adaptación de las Tareas XP

La aplicación de las tareas XP busca lograr con éxito la alta comunicación con el cliente y la agilidad en el proceso de desarrollo.

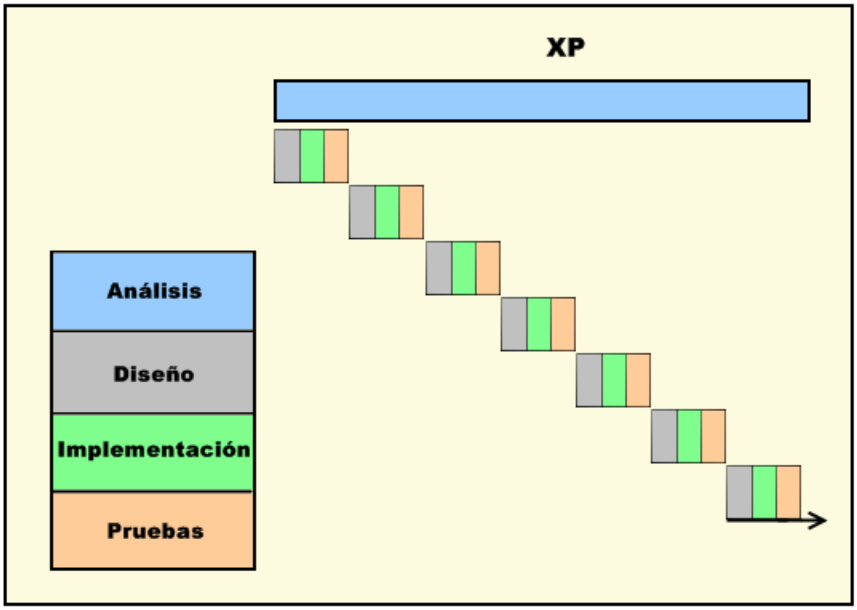


Figura Nº 4-1. Desarrollo en XP.

Se tienen cinco tareas fundamentales (ver figura 4-1) durante las iteraciones las cuales serán adaptadas de la siguiente manera:

- **Objetivo:** Muestra una breve descripción de los requerimientos de software a ser desarrollados en cada iteración. En la presente adaptación del proceso de desarrollo, se plantea esta actividad no perteneciente al modelo XP, la cual permite ampliar y describir más a fondo las funcionalidades que deben ser implementadas en dicha iteración.

- **Planificación:** Para la planificación se utilizan las historias de usuario, utilizando el formato de plantilla definido en la tabla 4-2.

- **Diseño:** En el diseño se elaboran las definiciones y los diagramas, como el diagrama de tablas de la base datos o algún otro diagrama que permita comprender y solucionar el problema.

- **Codificación:** En la presente adaptación del proceso de desarrollo se extraerán y mostrarán partes del código fuente que sean esenciales en la comprensión del sistema y la solución a los requerimientos de las historias de usuario.

- **Pruebas:** Las pruebas serán de aceptación en donde el usuario o cliente pone a prueba el sistema y verifica que hayan quedado cubiertos los requerimientos expresados en la historia de usuario correspondiente.

Es posible que en algunas iteraciones no se tengan desarrolladas todas las cinco tareas. Esto se debe a las actividades que involucran determinadas historias de usuario durante una iteración.

4.1.5 Actores y Responsabilidades

Existen diferentes roles (actores) y responsabilidades en XP para diferentes actividades y propósitos durante el proceso. Para este trabajo especial de grado los roles existentes son:

a. **Desarrollador:** quien es el responsable de tomar las decisiones técnicas y de llevar a cabo la codificación, el diseño y realizar pruebas al software.

b. **Cliente:** quien es parte del equipo, determina qué construir y cuándo, desarrolla pruebas funcionales del software para determinar cuando está completo un determinado aspecto.

En la tabla 4-3 se muestran las personas encargadas para cada rol.

Rol	**Nombre**
Desarrollador	Br. María Eugenia Márquez
Desarrollador	Br. Wilmer Fernández
Cliente	Prof. Sergio Rivas
Cliente	Profa. Jossie Zambrano

Tabla Nº 4-3. Roles existentes durante el desarrollo.

4.2 Metáfora del Sistema

Un lenguaje de metáforas se utiliza para describir la arquitectura del sistema. Esto ayuda a la comunicación entre las personas involucradas en el desarrollo del sistema.

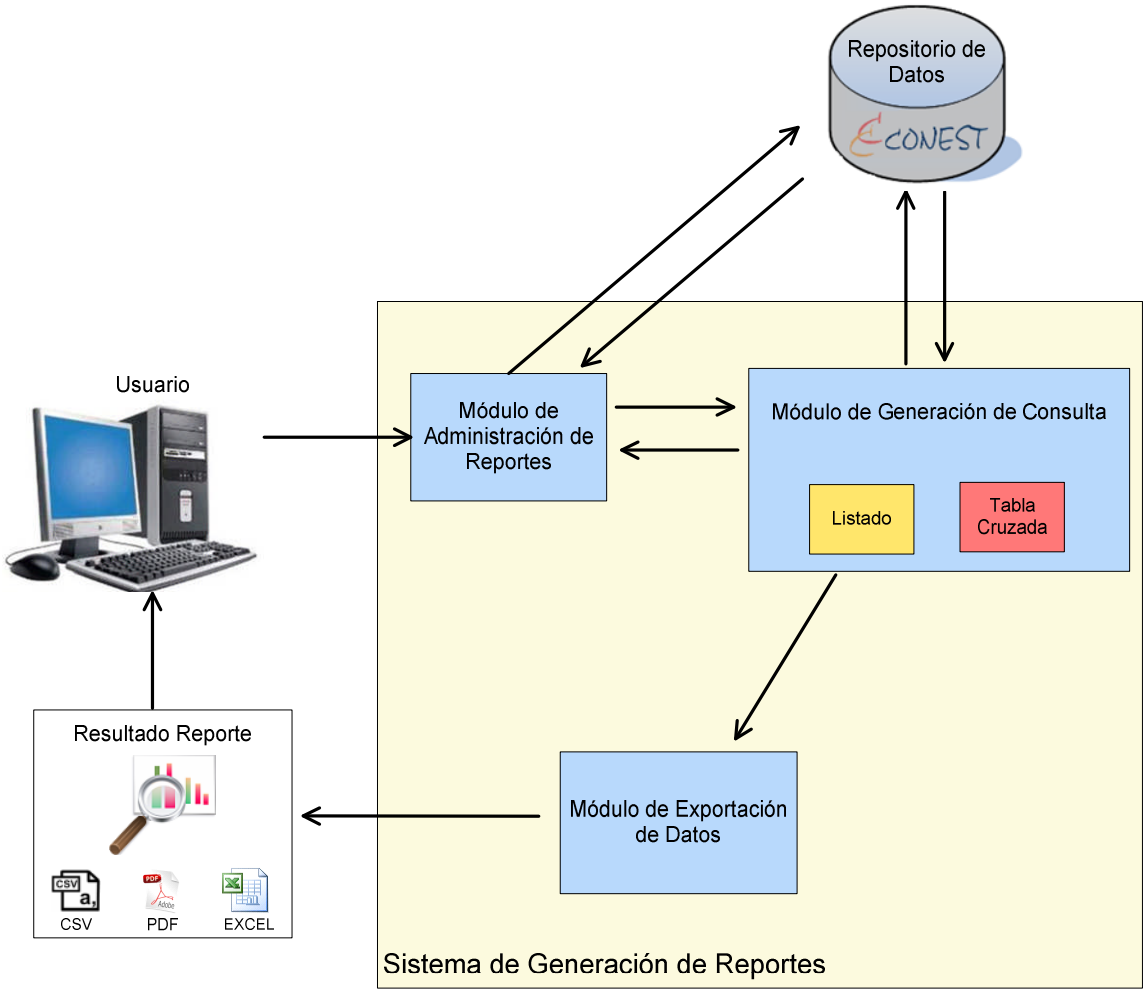


Figura Nº 4-2. Metáfora del sistema.

A continuación se describen los módulos principales que conformarán el sistema a desarrollar:

a. **Módulo de "Administración de Reportes":** permite el manejo de los reportes creados por los usuarios. Este módulo se encarga del almacenamiento y búsqueda de reportes, además incluye el manejo de los permisos de acceso de los reportes creados por los diversos usuarios que tienen acceso al sistema.

b. **Módulo de "Generación de Consulta":** permite al usuario seleccionar los atributos de las entidades que desea consultar y generar con estos datos un reporte, todo mediante secuencias de pasos específicamente diseñadas para cada uno de los tipos de reportes (listado y tabla cruzada).

c. **Módulo de "Exportación de Datos":** este módulo se encarga de generar los reportes solicitados en los tipos de documentos soportados por el sistema (PDF, EXCEL y CSV), a partir de los datos obtenidos de la consulta elaborada.

4.3 Requerimientos Generales del Sistema

La aplicación que se va a desarrollar es fundamentalmente un módulo para el sistema CONEST que permita la generación de reportes de los datos almacenados en su repositorio de datos.

Los usuarios deben tener experiencia en el manejo de aplicaciones ofimáticas y navegadores Web para utilizar el sistema. Para el manejo de algunas opciones, será necesario conocimientos básicos sobre el lenguaje de consulta SQL.

Se tiene especial cuidado en desarrollar una aplicación incrementable y escalable, dado al cambiante modelo de datos del sistema CONEST con el que el sistema debe interactuar.

4.3.1 Requerimientos Funcionales

Entre los requerimientos funcionales más notables del sistema de generación de reportes tenemos los siguientes:

- Generar reportes y simplificar dicha tarea mediante el establecimiento de secuencias de pasos (wizard).
- Permitir al usuario obtener el resultado del reporte en tipos documentos que puedan ser visualizados en aplicaciones ofimáticas e impresos. Los formatos soportados son PDF, EXCEL y CSV.
- Brindar al usuario la capacidad de generar los reportes en base a plantillas con características de formatos predefinidos.
- Ofrecer al usuario la opción de guardar los reportes y sus características, para posterior obtención de los resultados.

4.3.2 Requerimientos No Funcionales

Los requerimientos no funcionales especifican propiedades del sistema como restricciones de ambiente y desarrollo, rendimiento, dependencias de plataformas, mantenibilidad y confiabilidad. Los requerimientos de rendimiento imponen condiciones sobre los requerimientos funcionales como velocidad, tiempo de respuesta y uso de la memoria. Para este tipo de sistema los requerimientos no funcionales identificados son los siguientes:

- **Entorno de desarrollo.**
  - Se deben manejar las herramientas necesarias para que de manera integrada apoyen el ciclo de vida del proyecto, como herramientas de modelamiento visual, control de versiones, etc.

- **Desempeño.**
  - Garantizar la confiabilidad, disponibilidad, seguridad y el desempeño del sistema a los usuarios mediante la utilización de una plataforma robusta, eficiente y escalable.
  - Los datos almacenados podrán ser consultados y actualizados permanente y simultáneamente.
  - El sistema debe estar en capacidad de dar respuesta al acceso de todos los usuarios con tiempo de respuesta aceptable.

- **Escalabilidad.**
  - El sistema debe ser construido sobre la base de un desarrollo evolutivo e incremental, de manera tal que nuevas funcionalidades y requerimientos relacionados puedan ser incorporados afectando el código existente de la menor manera posible; para ello deben incorporarse aspectos de reutilización de componentes.

- **Facilidad de Uso.**
  - El sistema debe ser de fácil uso para los usuarios, presentar mensajes de error claros y concisos que permitan al usuario identificar el tipo de error y hacer uso de elementos visuales (metáforas) claras y de acuerdo al contexto.

- **Seguridad.**
  - El acceso al sistema debe estar restringido por el uso de claves asignadas a cada uno de los usuarios. Sólo podrán ingresar las personas que estén autenticadas debidamente en el sistema CONEST.
  - Debe contar con facilidades para administración de reportes que permita definir restricciones y acciones permitidas para cada usuario o grupo de usuarios.

- **Arquitectura.**
  - El sistema planteado es una aplicación Web, por tanto la solución debe operar de manera independiente del navegador que se utilice.

Estos requerimientos están cubiertos por las tecnologías referenciadas en la parte I del presente trabajo, donde se plantea la propuesta de Trabajo Especial de Grado.

CAPÍTULO 5

DESARROLLO DE LA APLICACIÓN

Este capítulo trata sobre el desarrollo de la aplicación y se encuentra dividido en iteraciones, durante las cuales se implementan las soluciones a los requerimientos.

5.1 Iteración 0

5.1.1 Objetivo

Se instalan aplicaciones y librerías con el fin de tener un ambiente de trabajo, también se identifican las estructuras a utilizar del modelo de CONEST y se define el modelo de datos que permita almacenar la información de los reportes creados por los usuarios en el repositorio de datos.

5.1.2 Planificación

En la tabla 5-1 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
1.00	08/10/2008	Definir el modelo de datos para el repositorio de datos del sistema e identificar estructuras necesarias del modelo de CONEST.	Nueva
2.00	08/10/2008	Definir los formatos de los documentos XML necesarios a ser almacenados.	Nueva

Tabla Nº 5-1. Historias de usuario. Iteración 0.

5.1.3 Diseño

En la figura 5-1 se muestra el diagrama entidad-relación de las estructuras de datos del sistema. En ella se pueden apreciar los datos que se almacenan de los reportes y las interrelaciones entre las nuevas estructuras persistentes del sistema con algunas estructuras preexistentes en el modelo del sistema CONEST.

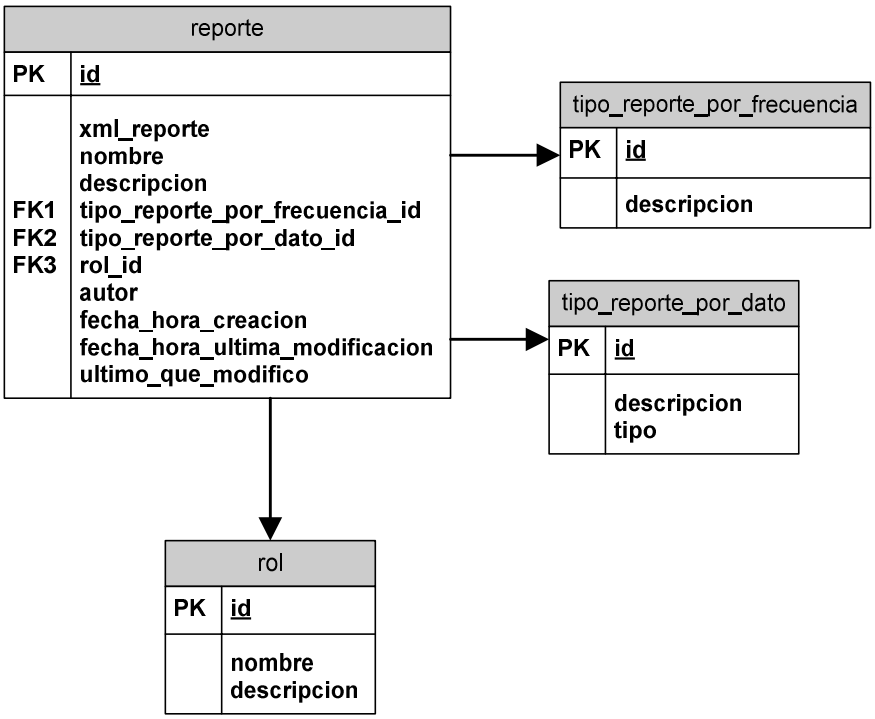


Figura Nº 5-1. Diagrama entidad-relación del sistema.

El modelo de datos planteado está compuesto por cuatro tablas, donde la tabla rol pertenece al modelo de datos de CONEST y las tablas reporte, tipo_reporte_por_frecuencia y tipo_reporte_por_dato es necesario crearlas para la persistencia de los reportes en el repositorio de datos. En la tabla reporte encontramos un campo denominado xml_reporte, que va a almacenar todos los datos relacionados con el reporte que el usuario elabore. En la figura 5-2 se puede apreciar la estructura del campo xml_reporte.

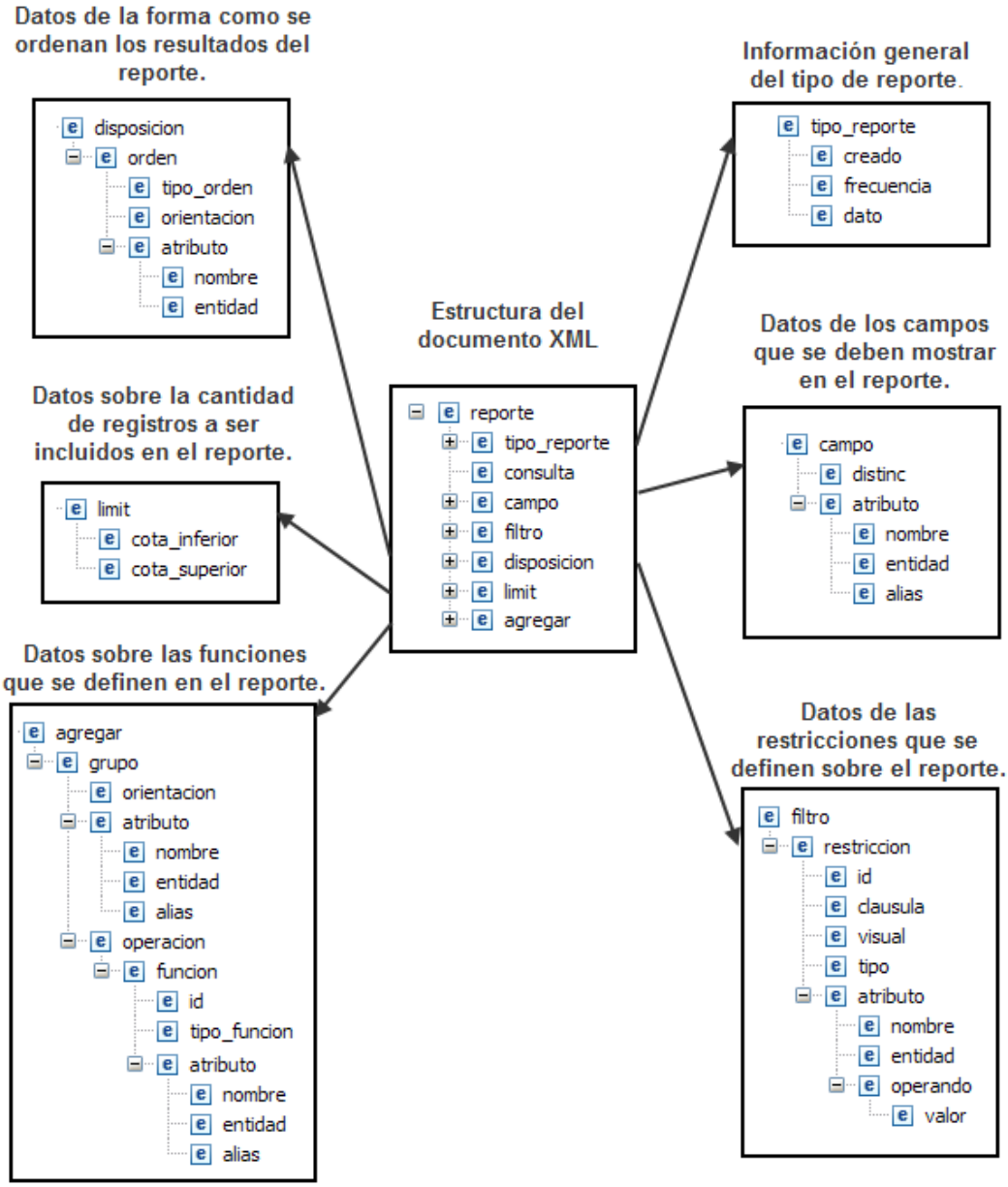


Figura Nº 5-2. Estructura del campo xml_reporte.

5.2 Iteración 1

5.2.1 Objetivo

En esta iteración se crean las primeras versiones de las interfaces, se comienza con el desarrollo de algunas funcionalidades básicas, como el listado de reportes que se encuentran en el repositorio de datos y un buscador de reportes. También se desarrollan las funcionalidades del lado del cliente del primer paso del wizard, en el que se le solicita al usuario que indique algunos datos del reporte, como el nombre, la descripción, la forma como desea realizar el reporte (a través de una secuencia de pasos o introduciendo una consulta en lenguaje SQL), entre otros.

5.2.2 Planificación

En la tabla 5-2 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
3.00	22/10/2008	Desarrollo de funcionalidades de Administración de Reportes.	Nueva
4.00	22/10/2008	Elaboración del paso 1 del wizard.	Nueva

Tabla Nº 5-2. Historias de usuario. Iteración 1.

5.2.3 Diseño

Las clases necesarias para la realización de los requerimientos se muestran en la figura 5-3.

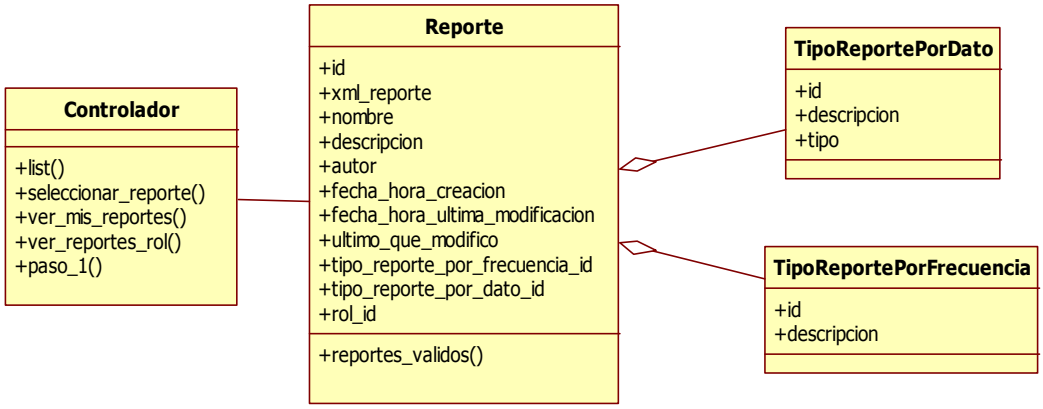


Figura Nº 5-3. Diagrama de clases del sistema.

Las interfaces siguen un patrón similar debido a que las vistas utilizan una misma plantilla o layout. En la figura 5-4 se puede apreciar la interfaz que muestra el listado de reportes que están guardados en el repositorio de datos y el buscador de reportes.

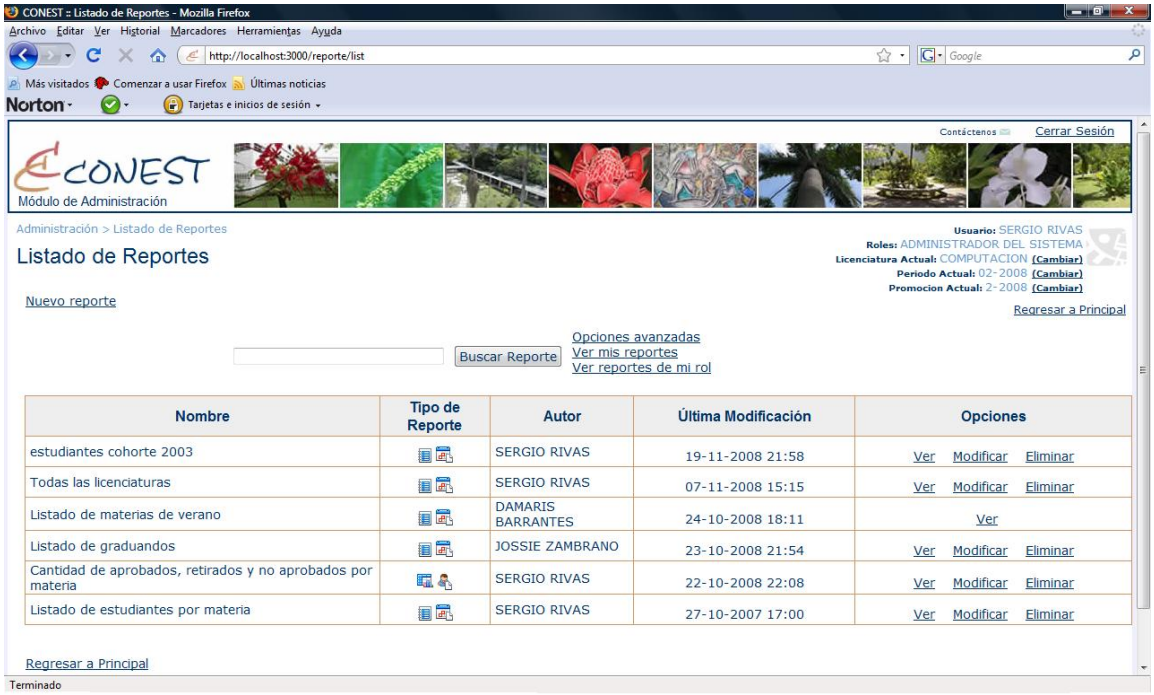


Figura Nº 5-4. Listado de reportes.

5.2.4 Codificación

En el fragmento de código de la figura 5-5 se muestra parte de la implementación del método que hace la búsqueda de un reporte en el repositorio de datos.

```
324     sql = "SELECT * FROM reporte WHERE "
325     if(!checked_nombre.empty? && checked_autor.empty? && checked_descripcion.empty?)
326         sql = sql << "reporte.nombre like '%" << nombre.strip << "%' "
327     elsif(!checked_autor.empty? && checked_nombre.empty? && checked_descripcion.empty?)
328         sql = sql << "reporte.autor like '%" << nombre.strip << "%' "
329     elsif(!checked_descripcion.empty? && checked_nombre.empty? && checked_autor.empty?)
330         sql = sql << "reporte.descripcion like '%" << nombre.strip << "%' "
331     elsif(!checked_nombre.empty? && !checked_autor.empty? && checked_descripcion.empty?)
332         sql = sql << "reporte.nombre like '%" << nombre.strip << "%'
333         OR reporte.autor like '%" << nombre.strip << "%' "
334     elsif(!checked_nombre.empty? && !checked_descripcion.empty? && checked_autor.empty?)
335         sql = sql << "reporte.nombre like '%" << nombre.strip << "%'
336         OR reporte.descripcion like '%" << nombre.strip << "%' "
337     elsif(!checked_autor.empty? && !checked_descripcion.empty? && checked_nombre.empty?)
338         sql = sql << "reporte.autor like '%" << nombre.strip << "%'
339         OR reporte.descripcion like '%" << nombre.strip << "%' "
340     elsif((!checked_nombre.empty? && !checked_autor.empty? && !checked_descripcion.empty?) ||
341           (checked_nombre.empty? && checked_autor.empty? && checked_descripcion.empty?))
342         sql = sql << "reporte.nombre like '%" << nombre.strip << "%'
343         OR reporte.autor like '%" << nombre.strip << "%'
344         OR reporte.descripcion like '%" << nombre.strip << "%' "
345     end
346     sql = sql << "ORDER BY reporte.#{ordenar_por};"
347     @reporte = Reporte.find_by_sql(sql)
```

Figura Nº 5-5. Código de búsqueda de un reporte.

Este método básicamente consiste en obtener los datos del reporte ingresados por el usuario en el campo de texto del buscador y las opciones de búsqueda avanzada, las cuales son provistas a través de elementos de selección. Una vez obtenido esto se construye la consulta en lenguaje SQL y se invoca al método find_by_sql (línea 347) el cual va a retornar un arreglo de reportes que cumplan con dichos criterios. En caso de que el usuario no ingrese datos en el campo de texto del buscador, se le mostrará un mensaje de error indicándole que debe ingresar datos del reporte.

El próximo extracto de código (figura 5-6) muestra cómo se realiza la búsqueda de reportes cuyo autor sea el usuario autenticado en la sesión actual, donde básicamente lo que se hace es obtener el nombre del usuario que está autenticado y se construye una consulta en lenguaje SQL (línea 358) donde el campo autor de los reportes contenga dicho nombre.

```
354  def ver_mis_reportes
355
356    autor= session[:usuario].nombre_estandar
357    @titulo_pagina='Reportes del usuario ' << autor
358    sql = "SELECT * FROM reporte WHERE reporte.autor ='" << autor << "' ORDER BY reporte.fecha_hora_ultima_modificacion DESC;"
359    r = Reporte.find_by_sql(sql)
360    if (r.empty?)
361        @error='Usted no tiene reportes'
362        redirect_to :action => 'busqueda', :error => @error
363    else
364        @reporte=r
365    end
366  end
```

Figura Nº 5-6. Código de la opción Ver mis reportes.

En el segmento de código que se muestra a continuación (figura 5-7), se puede observar la búsqueda de reportes asociados al rol del usuario actual. Lo primero que se obtiene es el rol o roles del usuario autenticado en la sesión actual y en base a estos roles se construye la consulta, donde el campo rol del reporte sea igual a alguno de los roles del usuario en el sistema.

```
367  def ver_reportes_rol
368    roles= session[:usuario].todos_roles
369    @titulo_pagina='Reportes del rol '
370    sql = "SELECT * FROM reporte WHERE "
371    i=0
372    for r in roles
373        if (i==0)
374            sql= sql << "reporte.rol_id = '"<< r.id << "'"
375        else
376            sql= sql << "OR reporte.rol_id = '"<< r.id << "'"
377        end
378        i=i+1
379    end
380    sql = sql << "ORDER BY reporte.fecha_hora_ultima_modificacion DESC;"
381    r = Reporte.find_by_sql(sql)
382    if (r.empty?)
383        @error='Este rol no tiene reportes'
384        redirect_to :action => 'busqueda', :error => @error
385    else
386        @reporte=r
387    end
388  end
```

Figura Nº 5-7. Código de la opción Ver reportes de mi rol.

5.2.5 Pruebas

La interfaz diseñada resultó sobrecargada. Se recomienda separar el buscador de reportes del listado.

Para probar la búsqueda de reportes se ingresaron datos en el campo de texto del buscador y se seleccionaron algunos criterios de las opciones avanzadas y se obtuvieron los resultados esperados.

5.3 Iteración 2

5.3.1 Objetivo

Se desarrolla la funcionalidad de manejo del reporte en la sesión y la opción SQL de la secuencia de pasos, en la que se evalúa la validez de una consulta ingresada por el usuario y en base a la ejecución de esta se obtienen los resultados del reporte. También se implementó la funcionalidad de almacenamiento de reportes en el repositorio de datos.

5.3.2 Planificación

En la tabla 5-3 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
4.00	05/11/2008	Manejo del reporte en la sesión del usuario.	Nueva
5.00	05/11/2008	Desarrollo de la opción SQL de la secuencia de pasos.	Nueva
6.00	05/11/2008	Almacenamiento del reporte en el repositorio de datos.	Nueva

Tabla Nº 5-3: Historias de usuario. Iteración 2.

5.3.3 Diseño

Los métodos necesarios para elaborar los requerimientos son incorporados a la clase controlador y se pueden observar en la figura 5-8.

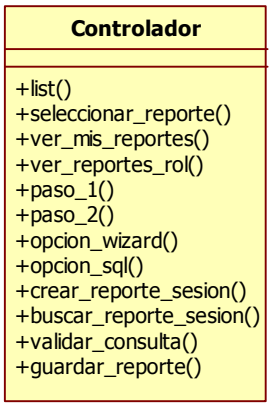


Figura Nº 5-8. Diagrama de la clase Controlador.

El método paso_2 se encarga de invocar a crear_reporte_sesion, el cual crea un objeto reporte y lo coloca en un hash de objetos reportes que se encuentra en la sesión, retornando un identificador o índice para tener acceso a dicho reporte cuando se desee, luego invoca al método buscar_reporte_sesion, el cual busca en el hash de reportes de la sesión el reporte que posea el índice recibido por parámetro, posteriormente se guarda en el objeto Reporte los datos ingresados por el usuario en el paso 1 y dependiendo de la opción que escogió el usuario para elaborar el reporte, invoca al método opcion_wizard u opcion_sql.

El método opcion_sql, se encarga de buscar el reporte en la sesión y de asignarle “Listado” como tipo de dato y posteriormente redirecciona a una vista donde se solicita al usuario la consulta en lenguaje SQL asociada al reporte. El método opcion_wizard, igualmente busca el reporte en la sesión y redirecciona a una interfaz donde el usuario debe escoger el tipo de dato del reporte: listado o tabla cruzada.

El método validar_consulta se encarga de evaluar la consulta ingresada por el usuario y determinar si es correcta sintácticamente o no, por otro lado, guardar_reporte se encarga de guardar en el repositorio de datos el reporte que se encuentra en la sesión.

También es necesario incorporar dos nuevos métodos a la clase Reporte: cargar_xml, el cual retorna un objeto XML (haciendo uso de la librería REXML) correspondiente al campo xml_reporte y cargar_consulta_formatoSQL el cual se encarga de guardar la consulta en el campo xml_reporte. En la figura 5-9 se pueden observar dichos métodos en la clase Reporte.

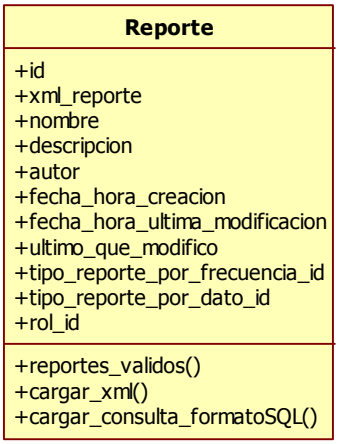


Figura Nº 5-9. Diagrama de la clase Reporte.

5.3.4 Codificación

El código del método crear_reporte_sesión se muestra en la figura 5-10. Este método recibe como parámetro la variable tipo_parametro, la cual posee el valor -1 cuando se está elaborando un reporte nuevo o posee como valor el id del reporte cuando este es modificado.

```
1262  def crear_reporte_sesion (tipo_parametro)
1263      if(tipo_parametro == -1) #Es un reporte totalmente nuevo
1264          xmlDoc= Document.new
1265          xmlDoc.add_element("reporte")
1266          elemento = ""
1267          #Agregar tipo
1268          tipo = Element.new("tipo_reporte")
1269          tipo2 = Element.new("creado")
1270          tipo3 = Element.new("frecuencia")
1271          tipo4 = Element.new("dato")
1272          tipo << tipo2
1273          tipo << tipo3
1274          tipo << tipo4
1275          xmlDoc.root.add_element(tipo)
1276          report = Reporte.new
1277          xmlDoc.write(elemento, -1, false, false)
1278          report.xml_reporte = elemento
1279      else #Es un reporte que está guardado en el repositorio de datos
1280          report = Reporte.find(tipo_parametro)
1281      end
1282      if(report.nil?)
1283          @error = "Ocurrió un error. El reporte es nulo."
1284          error(@error)
1285      else
1286          indice = Time.now.to_i
1287          indice = indice.to_s
1288          if(session[:reportes].nil?)
1289              reportes = Hash.new
1290              session[:reportes] = reportes
1291              reportes.merge!({indice => report})
1292          else
1293              r = session[:reportes]
1294              r.merge!({indice => report})
1295          end
1296          return indice
1297      end
1298  end
```

Figura Nº 5-10. Código del método crear_reporte_sesion.

En caso de que sea un reporte nuevo, se crea un objeto XML cuya etiqueta inicial se llama reporte, a esta se le incorpora una etiqueta hija denominada tipo_reporte, la cual a su vez tiene como hijas las etiquetas creado, frecuencia y dato. Posteriormente, en la línea 1277, el método write del objeto XML genera una cadena de caracteres que se corresponde con el objeto XML y esta se guarda en el campo xml_reporte del objeto reporte creado. Si el reporte se encuentra almacenado en el repositorio de datos, se busca en base al id recibido.

Una vez obtenido el reporte se crea el índice o clave que será su apuntador en el hash de reportes de la sesión y se verifica si dicho hash existe. En caso de que no exista, se crea y a través del método merge! se inserta el objeto reporte con su apuntador.

Un fragmento del código del método validar_consulta se puede apreciar en la figura 5-11, en el que se ejecuta la consulta ingresada por el usuario y es guardada en la etiqueta correspondiente del campo xml_reporte, en caso de que la consulta no arroje resultados se le notifica al usuario, de lo contrario este es redireccionado a una interfaz donde se muestra la opción de exportación del resultado del reporte a los formatos soportados por la aplicación. En la línea 772

se puede observar la captura de una excepción, esto es necesario realizarlo debido a que el usuario puede ingresar una consulta cuya sintaxis sea incorrecta y se le notifica a través de un mensaje de error.

```
737             begin
738                 mysql = ActiveRecord::Base.connection
739                 consulta1 = consulta.dup << ";"
740                 results = mysql.execute(consulta1)
741                 xmlDoc= report.cargar_xml
742                 report.cargar_consulta_formatoSQL(xmlDoc, consulta)
743                 report.descargar_xml(xmlDoc)
744                 if (results.num_rows == 0) then
745                     @error='La consulta no arroj&oacute; resultados'
746                     @cons = consulta
747                     @nombre_reporte = report.nombre
748                     @titulo_pagina='Paso 2 - Consulta del Reporte'
749                     @descripcion_paso = "Introduzca la consulta en lenguaje SQL que desea utilizar para el reporte."
750                     @paso = "2"
751                     @total = "3"
752                     @mostrar_reporte = true
753                     render :action => 'opcion_sql'
754                 else
755                     @cons= consulta
756                     @resultado=results
757                     @roles= session[:usuario].todos_roles
758                     if (@roles.size == 1)
759                         for rol in @roles
760                             @rol = rol.id
761                         end
762                     end
763                     @nombre_reporte = report.nombre
764                     @titulo_pagina='Paso 3 - Exportar Reporte'
765                     @descripcion_paso = "Descargue el resultado del reporte a través del enlace Exportar"
766                     @paso = "3"
767                     @total = "3"
768                     @mostrar_reporte = true
769                     @origen="sql"
770                     render :action => 'elegir_formato'
771                 end
772             rescue ActiveRecord::StatementInvalid => e
773                 @error='La consulta es inv&aacute;lida'
774                 @cons = consulta
775                 @nombre_reporte = report.nombre
776                 @titulo_pagina='Paso 2 - Consulta del Reporte'
777                 @descripcion_paso = "Introduzca la consulta en lenguaje SQL que desea utilizar para el reporte."
778                 @paso = "2"
779                 @total = "3"
780                 @mostrar_reporte = true
781                 render :action => 'opcion_sql'
782             end
```

Figura Nº 5-11. Código del método validar_consulta.

En la figura 5-12 se muestra un fragmento del código del método guardar_reporte, el cual busca en la sesión el reporte que posea el id recibido, se ingresan valores a los campos ultimo_que_modifico, fecha_hora_creacion y fecha_hora_ultima_modificacion, y finalmente se invoca al método save! (línea 1335) que se encarga de guardar el objeto Reporte en el repositorio de datos. En caso de que ocurra una excepción mientras se intenta guardar el reporte, esta es capturada en la línea 1342.

```
1329        report.ultimo_que_modifico = session[:usuario].nombre_estandar
1330        if(report.fecha_hora_creacion.nil?)
1331            report.fecha_hora_creacion = Time.now
1332        end
1333        report.fecha_hora_ultima_modificacion = Time.now
1334        begin
1335            if report.save!
1336              @mensaje = 'El reporte fue guardado exitosamente.'
1337              redirect_to :action => 'show', :id => report.id
1338            else
1339                @error = "Ocurrió un error. Reporte no se pudo guardar."
1340                error(@error)
1341            end
1342        rescue ActiveRecord::RecordInvalid => invalid
1343            mensaje = invalid.to_s
1344            if (mensaje.index('Nombre'))
1345                mensaje = 'Ya existe un reporte con ese nombre'
1346            end
1347            @error = 'El reporte no fue guardado por el siguiente error: ' << mensaje
1348            @roles= session[:usuario].todos_roles
1349            if (@roles.size == 1)
1350                for rol in @roles
1351                    @rol = rol.id
1352                end
1353            end
1354        end
```

Figura Nº 5-12. Código del método guardar_reporte.

5.3.5 Pruebas

Las pruebas de aceptación para la validación de la consulta consistieron en ingresar diversas consultas, algunas correctas y otras incorrectas sintácticamente, y los resultados obtenidos fueron los esperados.

5.4 Iteración 3

5.4.1 Objetivo

En esta iteración se realizó la exportación de resultados a formato CSV y se mejoró el modelo de datos implementando un conjunto de vistas para facilitar al usuario el acceso a los datos y se desarrollaron procedimientos almacenados (stored functions) para validar los datos nulos en las vistas.

5.4.2 Planificación

En la tabla 5-4 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
7.00	19/11/2008	Exportación de resultados a formato CSV.	Nueva
1.01	19/11/2008	Modelo de datos para el repositorio de datos del sistema.	Mejora

Tabla Nº 5-4: Historias de usuario. Iteración 3.

5.4.3 Diseño

La exportación de reportes la lleva a cabo la clase Exportador mostrada en la figura 5-13, la cual se encarga de manejar toda la lógica para exportar los resultados del reporte a los formatos de archivos soportados por el sistema (PDF, Excel, CSV).

Figura Nº 5-13. Diagrama de la clase Exportador.

Fue necesario crear el método exportar_reporte en la clase Controlador (figura 5-14) el cual recibe el tipo de archivo al que el usuario desea exportar el reporte y en base a esto resuelve

a cual método invocar; para esta iteración este método sólo invoca a exportar_csv de la clase Exportador.

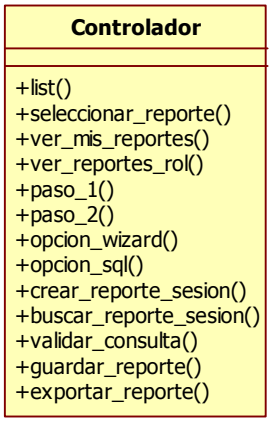


Figura Nº 5-14. Diagrama de la clase Controlador.

Por otro lado, era necesario mejorar el modelo de datos del sistema debido a que no se poseían estructuras que almacenaran los campos de las entidades explicadas en la parte II del presente trabajo, por ello se crearon vistas en el repositorio de datos. En la figura 5-15 se muestra un diagrama con las vistas creadas en el repositorio de datos.

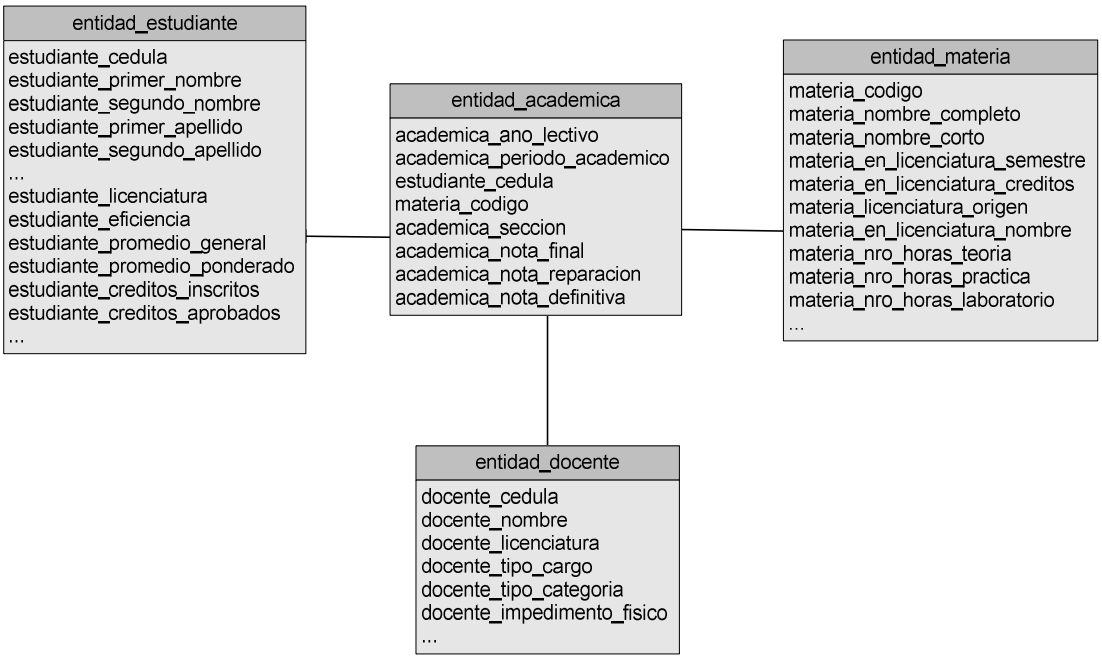


Figura Nº 5-15. Diagrama de las vistas creadas en el repositorio de datos.

5.4.4 Codificación

La implementación de la exportación de resultados a formato CSV se puede observar en la figura 5-16.

```
7   def exportar_csv(results, nombre, global)
8       nombre1 = nombre.gsub(' ', '_')
9       nombre1 = nombre1.dup << "_"<<(Time.now.to_i).to_s
10      FasterCSV.open("#{RAILS_ROOT}/public/reporte/#{nombre1}.csv", "w", {:force_quotes => true, :col_sep => ";"}) do |csv|
11          results.data_seek(0)
12          columnas = Array.new
13          results.fetch_fields.each do |info|
14              valor = convert_encoding(info.name)
15              columnas << valor
16          end
17          csv << columnas
18          results.each do |row|
19              fila = Array.new
20              row.each do |campo|
21                  valor1 = convert_encoding(campo.to_s)
22                  fila << valor1
23              end
24              csv << fila
25          end
26      end
27      mensaje = 'El reporte fue exportado exitosamente a formato CSV.'
28      return mensaje
29  end
```

Figura Nº 5-16. Código del método exportar_csv.

El código anterior consiste en tomar el nombre del reporte ingresado por el usuario y reemplazar los espacios en blanco por el carácter guión bajo, también conocido por su nombre en inglés como underscore (_), con la finalidad de que los usuarios no tengan inconvenientes al descargar los reportes en plataformas UNIX o LINUX (línea 8), luego a este nombre se le concatena un número que se obtiene a partir de la fecha y hora actual convertida a tipo de dato entero para identificar al reporte de forma unívoca (línea 9). Posteriormente en la línea 10 se abre el archivo en modo escritura y se indica la ruta donde se va a guardar. Se obtienen los nombres de las cabeceras de las columnas (líneas 13-16) y las celdas de las filas del objeto results (líneas 20-23) y para cada uno de estos se invoca al método convert_encoding, el cual se encarga de convertir los strings recibidos del repositorio de datos, que tienen codificación UTF8, y los convierte a la codificación latina ISO8859-1, para que las letras acentuadas o con tildes sean escritas correctamente, esto debido a que al visualizar el archivo CSV con la herramienta Microsoft Excel estas letras eran reemplazadas por otros caracteres. Finalmente estos valores son escritos en el archivo CSV, a través de los métodos provistos por la librería FasterCSV.

5.4.5 Pruebas

Las pruebas de aceptación para la exportación de reportes a formato CSV se realizaron verificando que los resultados mostrados en este tipo de archivo eran los mismos que los arrojados por consultas SQL sobre la base de datos.

5.5 Iteración 4

5.5.1 Objetivo

Se desarrolló la selección de campos y definición de restricciones en la secuencia de pasos (Pasos 3 y 4). Esta última sólo fue desarrollada del lado del cliente debido a que el manejo del lado del servidor es muy complejo e implementarlo abarcaría más del tiempo estipulado para esta iteración.

5.5.2 Planificación

En la tabla 5-5 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
8.00	03/12/2008	Desarrollo de pasos 3 y 4 del wizard.	Nueva

Tabla Nº 5-5: Historias de usuario. Iteración 4.

5.5.3 Diseño

Se agregaron tres métodos a la clase Controlador, los cuales se pueden observar en la figura 5-17. El método paso_3 obtiene el tipo de dato seleccionado por el usuario (listado o tabla cruzada) y lo guarda en el objeto reporte de la sesión y en su campo xml_reporte, en base al tipo de dato recibido se determina a cual método se debe invocar; para esta iteración sólo se implementó el método opcion_listado el cual se encarga de obtener las entidades y sus campos para que estas sean mostradas en la interfaz de selección de campos. También se implementó el método paso_4 el cual obtiene los campos seleccionados por el usuario en el paso anterior y los guarda en el xml y además obtiene todas las entidades con sus campos para mostrarlos en la vista de definición de restricciones.

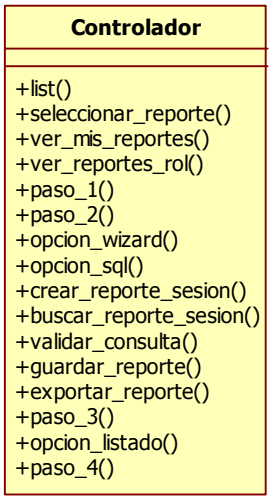


Figura Nº 5-17. Diagrama de la clase Controlador.

A la clase Reporte también se le incorporaron cuatro métodos más, los cuales se pueden observar en la figura 5-18. El método entidades_con_atributos se encarga de buscar en el repositorio de datos todos los campos de las vistas, cargar_campos_XML se encarga de colocar en el XML todos los campos que seleccionó el usuario, descargar_campos_XML retorna un hash que contiene los campos seleccionados por el usuario y descargar_xml se encarga de obtener el string correspondiente a un objeto XML dado y dicho string es guardado en el campo xml_reporte del objeto reporte.

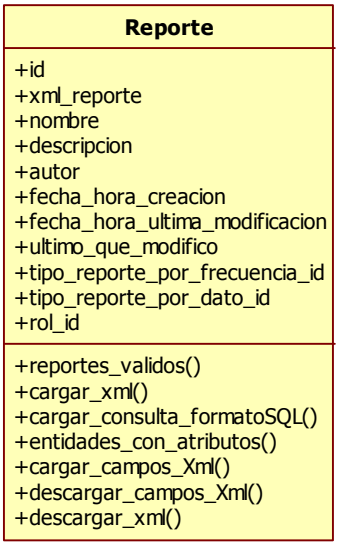


Figura Nº 5-18. Diagrama de la clase Reporte.

La interfaz de Selección de campos se puede observar en la figura 5-19. Esta interfaz posee en la parte izquierda tantas listas desplegables como vistas haya en el repositorio de datos y estas contienen todos sus campos. En la parte central se tienen dos botones para que el usuario agregue o elimine campos de la lista de la derecha y adicionalmente en esta parte se tienen dos botones para que el usuario ordene los campos seleccionados.

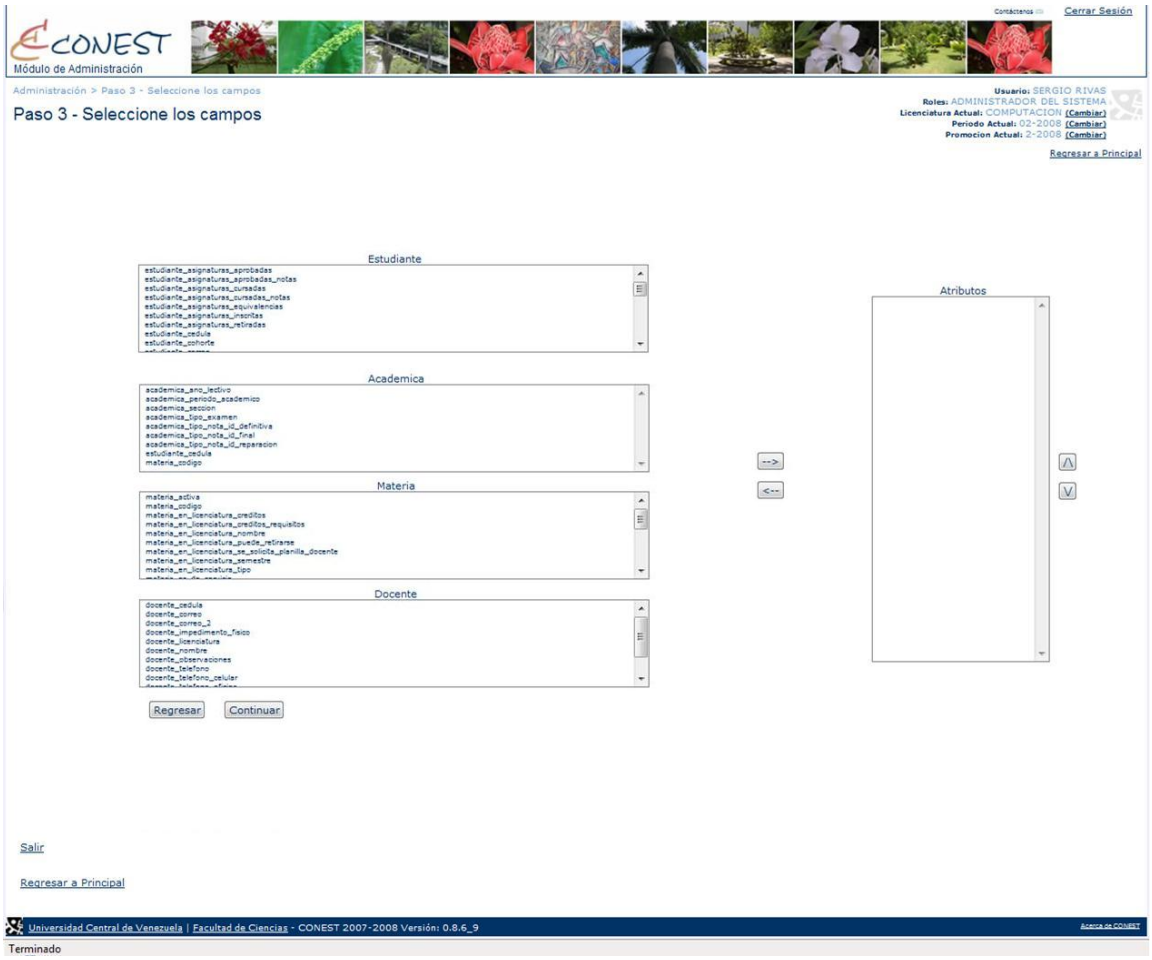


Figura Nº 5-19. Vista paso 3.

La interfaz de definición de restricciones se puede observar en la figura 5-20. Al igual que en la interfaz de selección de campos se muestran los campos de las entidades en la parte izquierda y además se muestra una lista desplegable que contiene los campos seleccionados en el paso anterior. Cuando el usuario selecciona algún campo de la izquierda y oprime el botón agregar (representado por una flecha que apunta hacia la derecha), se muestra en la parte inferior derecha unos campos para que el usuario defina el valor de la restricción. En la parte derecha de la interfaz

se muestran unos botones para que el usuario ordene las restricciones y para que agregue el conector OR o agrupe las restricciones a través de paréntesis.

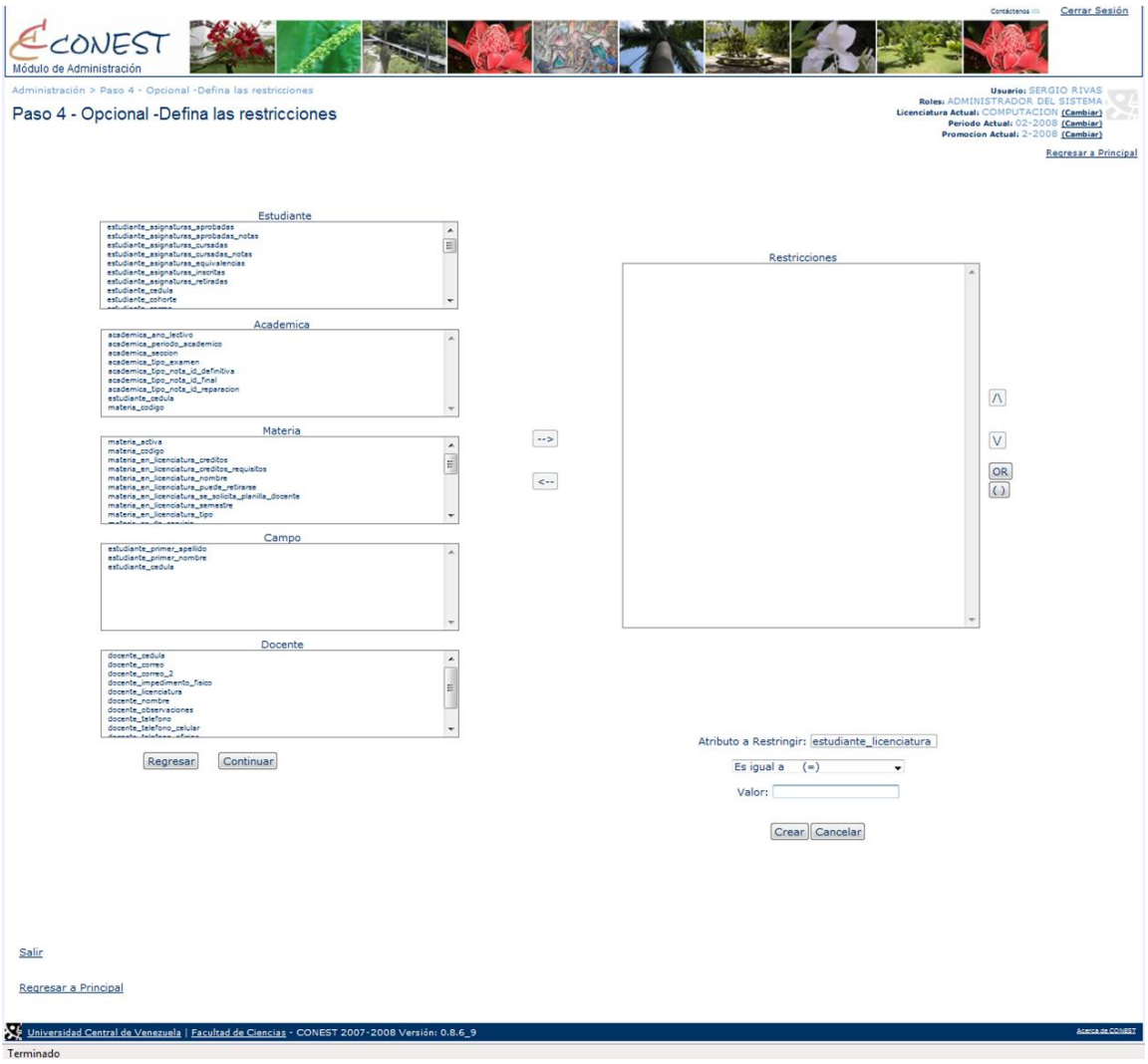


Figura Nº 5-20. Vista paso 4.

5.5.4 Codificación

En la figura 5-21 se muestra el código del método entidades_con_atributos, el cual es invocado tanto por el método opcion_listado como por el método paso_4 del objeto Controlador.

```
819  def entidades_con_atributos
820
821      database_name = ActiveRecord::Base.connection.current_database
822      mysql = ActiveRecord::Base.connection
823      entidades = mysql.execute("SHOW FULL TABLES WHERE Table_type = 'VIEW' AND
824                                 Tables_in_"<< database_name <<" like 'entidad_%' " << ";")
825      datos = Array.new
826      listas_atributos= Hash.new
827      if (entidades.num_rows > 0)
828          entidades.each do |row|
829              prueba= Array.new
830              nombre_entidad = row[0].split('_', 2)
831              listas_atributos.merge!({(nombre_entidad[1].capitalize!) => prueba})
832              results2 = mysql.execute("describe "<< row[0] << ";");
833              if (results2.num_rows > 0)
834                  results2.each do |datos|
835                      prueba.push(datos[0])
836                  end
837                  prueba.sort!
838              end
839          end
840      end
841      return listas_atributos
842  end
```

Figura Nº 5-21. Código del método entidades_con_atributos.

Como se puede observar en la línea 821, lo primero que se hace es obtener el nombre de la base de datos y en la línea 823 se ejecuta una consulta donde se solicitan todas las vistas de dicha base de datos, cuyos nombres comiencen con “entidad_”. En caso de que la consulta arroje resultados, para cada una de las entidades obtenidas se obtiene el nombre sin el prefijo “entidad_” y este nombre es el que va a servir como apuntador del arreglo de campos de dicha entidad, es decir, este método retorna un hash cuyas claves son los nombres de las entidades y estas apuntan a su arreglo de campos.

Luego, en la línea 832, se ejecuta una consulta que retorna, entre otras cosas, los campos de la entidad y estos son agregados al arreglo de campos. Una vez que se hayan recorrido todos los campos de la entidad este arreglo es ordenado y se repite el proceso con las demás entidades.

A continuación (figura 5-22) se muestra el código del método paso_4. Lo primero que se hace es obtener los campos seleccionados por el usuario (línea 617) y buscar el reporte en la sesión.

```
615  def paso_4
616
617      opcion = @params['lista']['atributos']
618      @indice = @params['indice']
619      reporte = buscar_reporte_sesion(@indice)
620      if (opcion.nil?)
621        #Error con la seleccion de campos del usuario
622        @error='Ha ocurrido un error con su seleccion'
623        opcion_listado
624        render :action => 'opcion_listado'
625      else
626          if(!reporte.nil?)
627              xmlDoc= reporte.cargar_xml
628              if (xmlDoc != nil)
629                  reporte.cargar_campos_Xml(opcion, xmlDoc)
630                  texto=""
631                  xmlDoc.write(texto, -1, false, false)
632                  opcion= reporte.descargar_campos_Xml(xmlDoc)
633                  reporte.descargar_xml(xmlDoc)
634              end
635              #El usuario seleccion la opcion de crear un reporte de listado
636              @titulo_pagina='Paso 4 - Opcional -Defina las restricciones'
637              #Buscar datos de las entidades en la BD
638              @listas_atributos=reporte.entidades_con_atributos
639              @listas_atributos.merge!({'Campo' => opcion})
640              #Agregando scripts javascript
641              @page_javascript_includes = ['restriccion']
642              render :action => 'restricciones'
643          end
644      end
645  end
```

Figura Nº 5-22. Código del método paso_4.

En la línea 627 se obtiene el XML que está guardado en el objeto reporte y en la línea 629 se invoca al método cargar_campos_XML el cual va a colocar en el XML los campos que seleccionó el usuario en la interfaz. En la línea 631 se obtiene el string correspondiente a este objeto XML, luego se invoca al método descargar_campos_XML el cual retorna un hash con los campos seleccionados por el usuario y posteriormente este objeto XML es guardado en el objeto reporte a través del método descargar_xml.

Finalmente en la línea 638 se invoca al método entidades_con_atributos, el cual retorna un hash con los campos de las vistas y a este se le incorpora el hash de campos seleccionados por el usuario (línea 639), es decir, el hash @listas_atributos va a contener tanto los campos de las vistas que están en el repositorio de datos, como los campos que seleccionó el usuario previamente, esto con la finalidad de que en la siguiente interfaz el usuario pueda definir restricciones sobre cualquiera de estos campos.

En la figura 5-23 se muestra el código del método cargar_campos_XML el cual recibe dos argumentos: los campos seccionados por el usuario y el objeto XML a donde serán incorporados estos campos.

```
122  def cargar_campos_Xml(lista,documento_xml_reporte)
123    #Carga en Documento-Xml-reporte los campos seleccionados en la interfaz del reporte
124    if (documento_xml_reporte!= nil)
125      if (lista.length > 0)
126        documento_xml_reporte.elements.delete_all('reporte/campo//atributo') #Borro los viejos
127        tipo = Element.new("campo")
128        lista.each { |valor|
129            vista = valor.split('_', 2)
130            atributo = Element.new("atributo")
131            nombre = Element.new("nombre")
132            nombre.text = valor.strip
133            entidad = Element.new("entidad")
134            entidad.text = vista.first.strip
135            atributo << nombre
136            atributo << entidad
137            tipo << atributo
138        }
139        documento_xml_reporte.root.add_element(tipo)
140      end
141      return documento_xml_reporte
142    else
143      return nil
144    end
145  end
```

Figura Nº 5-23. Código del método cargar_campos_XML.

En la línea 126 lo que se hace es eliminar las etiquetas atributo que ya existan en ese objeto XML, en la siguiente línea se crea la etiqueta campo y a continuación se itera sobre el hash de campos recibidos y se crean las etiquetas atributo y nombre, a esta última se le incorpora el nombre del campo, luego se crea la etiqueta entidad y se le incorpora el nombre. A la etiqueta atributo se le incorporan como etiquetas hijas nombre y entidad y a la etiqueta campo se le incorpora como hijo la etiqueta atributo. Este proceso se repite tantas veces como atributos haya seleccionado el usuario. Finalmente en la línea 139 se incorpora la etiqueta campo al documento XML.

5.5.5 Pruebas

Las pruebas de aceptación para las interfaces fueron las verificaciones de que estas tuvieran los campos correctos en las listas desplegables y que esos datos hayan sido enviados y tratados como se esperaba por la aplicación.

Para probar que en el XML se estaban guardando los datos correctamente, se hizo que el código tuviera algunas líneas que generaran salida en consola y se obtuvieron los resultados esperados.

5.6 Iteración 5

5.6.1 Objetivo

En esta iteración se mejoró el modelo de datos agregando dos entidades nuevas: Graduando y Sección. También se elaboró la funcionalidad de ordenamiento y agrupación del resultado en base a uno o varios campos.

5.6.2 Planificación

En la tabla 5-6 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
1.02	17/12/2008	Modelo de datos para el repositorio de datos del sistema.	Mejora
9.00	17/12/2008	Desarrollo del paso 5 del wizard.	Nueva

Tabla Nº 5-6: Historias de usuario. Iteración 5.

5.6.3 Diseño

Al modelo de datos se le incorporaron dos nuevas entidades: Graduando, la cual se relaciona con la entidad Estudiante y contiene toda la información concerniente a la graduación y a los graduandos como tal, y Sección, la cual relaciona a la entidad Académica con Docente, debido a que no existía anteriormente un nexo entre estas dos entidades. El nuevo modelo de datos se muestra en la figura 5-24.

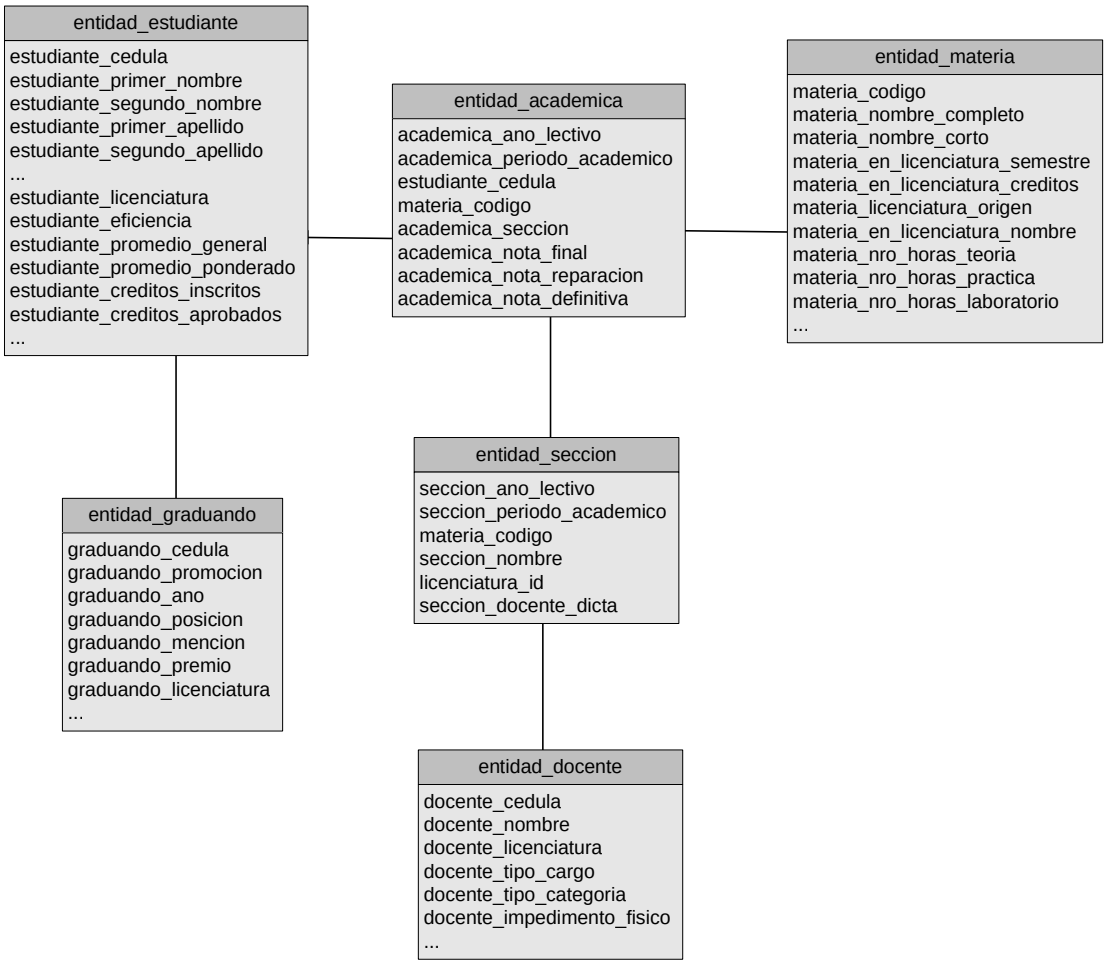


Figura Nº 5-24. Diagrama de las vistas incorporadas al repositorio de datos.

A la clase Controlador se le incorporaron dos nuevos métodos los cuales se pueden observar en la figura 5-25. El método paso_5 obtiene los campos seleccionados por el usuario en el paso 3 para que estos sean mostrados en la interfaz y el método paso_6 se encarga de obtener los campos en base a los cuales el usuario desea ordenar el resultado, los guarda en el XML y genera la consulta asociada a dicho reporte.

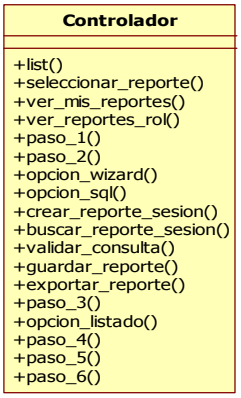


Figura Nº 5-25. Diagrama de la clase Controlador.

A la clase Reporte se le incorporaron cinco métodos para poder solventar los requerimientos de esta iteración (figura 5-26). El método cargar_ordenamiento_Xml se encarga de guardar los campos de orden en el XML, generar_consulta se encarga de invocar a los tres últimos métodos de esta clase para construir la consulta asociada al reporte: obtener_select_Xml_formatoSQL se encarga de obtener los campos seleccionados por el usuario y en base a ellos construye la cláusula select de la consulta, generar_enlaces_formatoSQL se encarga de construir los enlaces entre las vistas (inner join) involucradas en el reporte y obtener_ordenamiento_Xml_formatoSQL construye la cláusula order by de la consulta en base a los campos de ordenamiento indicados por el usuario. Para esta iteración no se implementó un método que obtuviera las restricciones debido a que aún no se ha desarrollado un método que haga el manejo de estas en el lado del servidor.

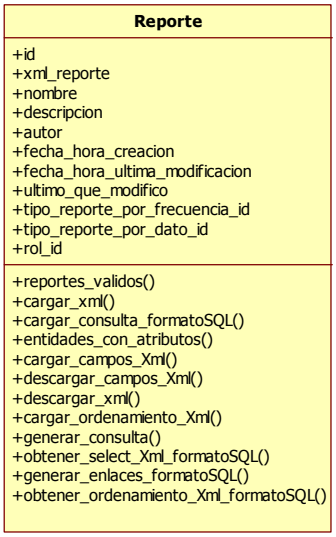


Figura Nº 5-26. Diagrama de la clase Reporte.

La interfaz del paso 5 de selección de orden se puede observar en la figura 5-27. En esta interfaz el usuario puede ordenar el resultado del reporte en base a los campos que seleccionó en el paso 3. En la parte derecha se ofrece un botón para que el usuario cambie el orden de ascendente a descendente y viceversa.

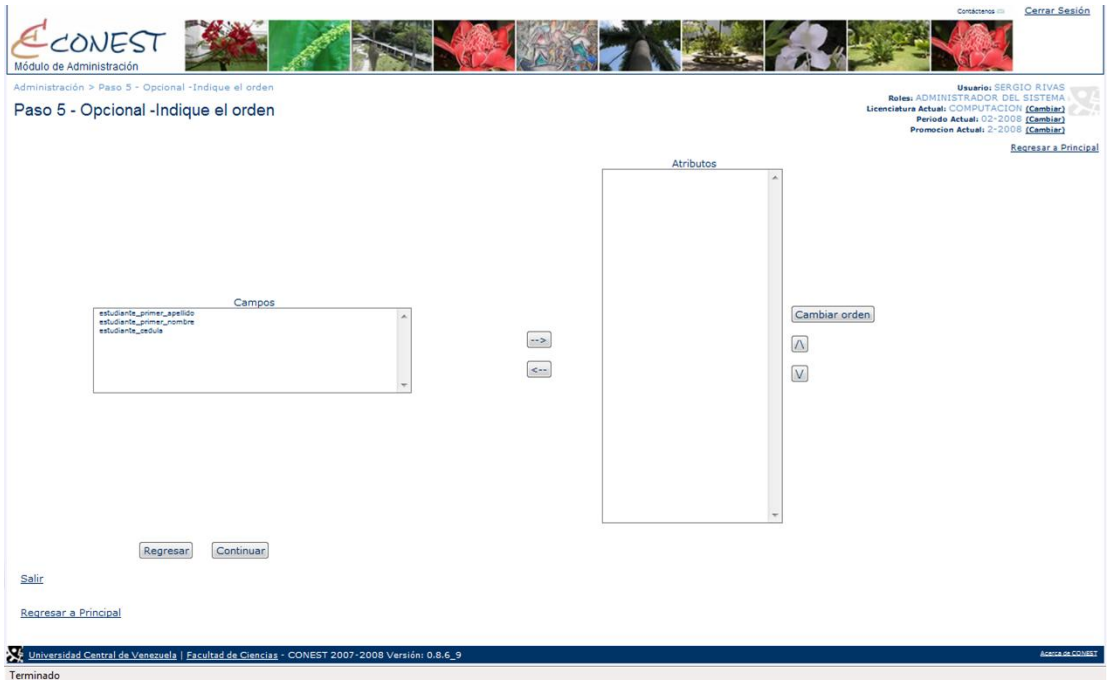


Figura Nº 5-27. Vista paso 5.

5.6.4 Codificación

En la figura 5-28 se puede observar el código del método paso_6, el cual se encarga de obtener los campos en base a los cuales el usuario desea ordenar el resultado del reporte (línea 680), en la línea 683 se obtiene el objeto XML asociado al reporte actual y en la línea 685 se invoca a un método que se encarga de incorporar al objeto XML los campos de ordenamiento seleccionados por el usuario, posteriormente en la línea 688 se invoca a un método que se encarga de construir la consulta asociada al reporte actual y luego esta es incorporada al objeto XML en la siguiente línea. Finalmente el objeto XML es guardado en el objeto reporte en la línea 690.

```
674  def paso_6
675
676      @indice = @params['indice']
677      reporte = buscar_reporte_sesion(@indice)
678      opcion = @params['lista']
679      if (!opcion.nil?)
680          opcion= @params['lista']['atributos']
681      end
682      if(!reporte.nil? && (!opcion.nil?))
683          xmlDoc= reporte.cargar_xml
684          if (xmlDoc != nil)
685              reporte.cargar_ordenamiento_Xml(xmlDoc,opcion)
686              #Generar la consulta
687              consulta_generada=""
688              consulta_generada=reporte.generar_consulta(xmlDoc)
689              reporte.cargar_consulta_formatoSQL(xmlDoc, consulta_generada)
690              reporte.descargar_xml(xmlDoc)
691              @titulo_pagina='Seleccione alguna de estas opciones'
692              @page_javascript_includes = ['orden']
693              @roles= session[:usuario].todos_roles
694              if (@roles.size == 1)
695                for rol in @roles
696                  @rol = rol.id
697                end
698              end
699              render :action => 'elegir_formato'
700          end
701      end
702  end
```

Figura Nº 5-28. Código del método paso_6.

A continuación (figura 5-29) se muestra el código del método cargar_ordenamiento_Xml, el cual recibe por parámetros el XML del reporte actual y los campos que el usuario escogió para realizar el ordenamiento. Lo primero que se hace es eliminar los campos de ordenamiento anteriores que haya definido el usuario (línea 169). Luego se crea la etiqueta disposición y se recorren los campos. La variable tipo_orden indica si el ordenamiento es ascendente o descendente y en campo_orden queda el nombre del campo incluyendo el nombre de la entidad. Posteriormente son creadas las etiquetas orden y tipo_orden, asignándole a esta última el valor contenido en la variable tipo_orden. En la línea 181 se separa el nombre de la entidad del nombre del campo, se crean las etiquetas atributo y nombre y a esta última se le asigna el nombre del campo, luego se crea la etiqueta entidad y se le asigna el nombre respectivo. Posteriormente a la etiqueta atributo se le asigna como hijas las etiquetas nombre y entidad, a la etiqueta orden se le asigna como hijas tipo_orden y atributo, a la etiqueta disposición se le asigna como hija orden y este proceso se repite para todos los campos que haya seleccionado el usuario. Finalmente la etiqueta disposición se incorpora al XML.

```
def cargar_ordenamiento_Xml (documento_xml_reporte, lista)
  #Carga en Xml-reporte los campos sobre los cuales se define un ordenamiento
  valor=""
  if (documento_xml_reporte!= nil)
      if (lista.length > 0)
          documento_xml_reporte.elements.delete_all('reporte/disposicion//orden') #Borro los viejos
          tipo = Element.new("disposicion")
          lista.each { |valor|
              vista = valor.split("Ordenar", 2)
              if (vista.length == 2)
                  tipo_orden= vista.pop
                  campo_orden= vista.pop
                  if (tipo_orden != nil)
                      orden = Element.new("orden")
                      ordenamiento = Element.new("tipo_orden")
                      ordenamiento.text = tipo_orden.strip
                      if (campo_orden != nil)
                          campo_orden_valores = campo_orden.split('_', 2)
                          atributo = Element.new("atributo")
                          nombre = Element.new("nombre")
                          nombre.text = campo_orden.strip
                          entidad = Element.new("entidad")
                          entidad.text = campo_orden_valores.first.strip
                          atributo << nombre
                          atributo << entidad
                          orden << ordenamiento
                          orden << atributo
                          tipo << orden
                      end
                  end
              end
          }
          documento_xml_reporte.root.add_element(tipo)
      end
      return documento_xml_reporte
  else
      return nil
  end
end
```

Figura Nº 5-29. Código del método cargar_ordenamiento_Xml.

5.6.5 Pruebas

Las pruebas de aceptación para la generación de la consulta se hicieron comparando los resultados obtenidos a través de la aplicación contra consultas SQL ejecutadas en la base de datos, llegando a la conclusión de que la aplicación genera las consultas correctamente.

Para probar que en el XML se estaban guardando los datos correctamente, se hizo que el código tuviera algunas líneas que generaran salida en consola y se obtuvieron los resultados esperados.

5.7 Iteración 6

5.7.1 Objetivo

Se mejoró el modelo de datos creando las vistas mediante el uso de enlaces de tipo left join en lugar de inner join, además los procedimientos almacenados (stored functions) que ya estaban creados se colocaron como determinísticos, con la finalidad de mejorar los tiempos de respuesta. También se realizó la exportación de los resultados del reporte a formato PDF y Excel.

5.7.2 Planificación

En la tabla 5-7 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
1.03	14/01/2009	Modelo de datos para el repositorio de datos del sistema.	Mejora
10.00	14/01/2009	Exportación de resultados a formato PDF.	Nueva
11.00	14/01/2009	Exportación de resultados a formato Excel.	Nueva

Tabla Nº 5-7: Historias de usuario. Iteración 6.

5.7.3 Diseño

Para dar solución a los requerimientos de esta iteración fue necesario incorporar dos nuevos métodos a la clase Exportador (figura 5-30).

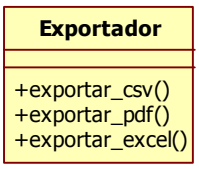


Figura Nº 5-30. Diagrama de la clase Exportador.

5.7.4 Codificación

En la figura 5-31 se muestra el código correspondiente al primer fragmento del método exportar_excel.

```
1601  def exportar_excel(results, nombre)
1602
1603    #Creacion del libro de trabajo
1604    nombre1 = nombre.gsub(' ', '_')
1605    nombre1 = nombre1.dup << "_"<<(Time.now.to_i).to_s
1606    spreadsheet_file = "#{RAILS_ROOT}/public/reporte/#{nombre1}.xls"
1607    workbook = Excel.new(spreadsheet_file)
1608    worksheet = workbook.add_worksheet
1609    results.data_seek(0)
1610    #Formatos
1611    table_header_format =Format.new(:size=> 12,
1612                                    :top => 2,
1613                                    :bottom => 2,
1614                                    :right => 2,
1615                                    :left => 2,
1616                                    :bold => 1,
1617                                    :color=>'black',
1618                                    :align => 'justify')
1619    data_format=Format.new(:align => 'justify',
1620                           :top => 1,
1621                           :bottom => 1,
1622                           :right => 1,
1623                           :left => 1)
1624    page_header_format = Format.new(:color=>'black', :bold=>true, :size=>12,:align => 'justify')
1625    title_format = Format.new(:color=>'black', :bold=>true, :size=>12,:align => 'center')
1626    workbook.add_format(table_header_format)
1627    workbook.add_format(page_header_format)
1628    workbook.add_format(data_format)
1629    workbook.add_format(title_format)
1630    # Cabecera de la Hoja de Excel
1631    current_row=-1
1632    current_column=-1
1633    worksheet.write_row((current_row += 1), 0, 'UNIVERSIDAD CENTRAL DE VENEZUELA', page_header_format)
1634    worksheet.write_row((current_row += 1), 0, 'FACULTAD DE CIENCIAS', page_header_format)
1635    worksheet.write_row((current_row += 1), 0, 'COORDINACION ACADEMICA', page_header_format)
```

Figura Nº 5-31. Código del primer fragmento del método exportar_excel.

En la línea 1604 se toma el nombre del reporte ingresado por el usuario y se reemplazan los espacios en blanco por el carácter guión bajo, también conocido por su nombre en inglés como underscore (_), con la finalidad de que los usuarios no tengan inconvenientes al descargar los reportes en plataformas UNIX o LINUX, luego a este nombre se le concatena un número que se obtiene a partir de la fecha y hora actual convertida a tipo de dato entero para identificar al reporte de forma unívoca. Posteriormente en la línea 1607 se crea el objeto Excel o libro de trabajo y se le pasa por parámetro la ruta donde se va a guardar dicho archivo.

En la línea 1608 se le agrega una hoja al libro de trabajo y luego se crean los formatos necesarios para las diferentes áreas del reporte, tales como la cabecera de la tabla (table_header_format), datos contenidos en la tabla (data_format), encabezado (page_header_format) y título del reporte (title_format) y estos se incorporan al libro de trabajo. En las líneas 1631 y 1632 se inicializan los apuntadores por fila y por columna respectivamente, y se escribe la cabecera del reporte en las filas y columnas indicadas con el formato establecido.

Posteriormente (figura 5-32) se obtienen los nombres de los campos a mostrar en el reporte y se escriben en el archivo (línea 1649). Luego se itera sobre el resultado del reporte y se van escribiendo celda por celda los valores (línea 1656), invocando previamente al método convert_encoding, el cual resuelve la incompatibilidad existente en la herramienta Microsoft Excel con los strings que tienen codificación UTF8 convirtiéndolos a la codificación latina ISO8859-1, para que las letras acentuadas o con tildes sean escritas correctamente, y finalmente en la línea 1661 se cierra el libro de trabajo.

```
1640    #Cabecera de la tabla de datos
1641    current_row += 1
1642    columns = Array.new
1643    columns << "Nro"
1644    index=0
1645    results.fetch_fields.each do |info|
1646        columns.push(info.name)
1647        index += 1
1648    end
1649    worksheet.write((current_row += 1),0, columns ,table_header_format)
1650    current_row += 1
1651    row_number=1
1652    results.each do |row|
1653        current_column = -1
1654        worksheet.write current_row, (current_column += 1), row_number, data_format
1655        row.each do |col|
1656            worksheet.write current_row, (current_column += 1), convert_encoding(col.to_s), data_format
1657        end
1658        current_row += 1
1659        row_number +=1
1660    end
1661    workbook.close
```

Figura Nº 5-32. Código del segundo fragmento del método exportar_excel.

A continuación (figura 5-33) se puede observar el fragmento inicial del código del método exportar_pdf.

```
211  def exportar_pdf(results, formato_pdf, nombre)
212
213      columnas = Array.new
214      results.fetch_fields.each do |info|
215          columnas << info.name.dup
216      end
217      pdf = PDF::Writer.new(:paper => "LETTER", :orientation => :portrait)
218      pdf.margins_cm(2, 3, 2, 2)
219      pdf.select_font "Times-Roman"
220      pdf.start_page_numbering(pdf.absolute_right_margin, (pdf.absolute_bottom_margin-(2*pdf.font_height(@font_size))),
221                               9, nil, pattern = "<PAGENUM>/<TOTALPAGENUM>", 1)
222      pdf.add_image_from_file("public/images/logo_ucv.jpg", pdf.absolute_left_margin, (pdf.absolute_top_margin-55),
223                               width = 60, height = 60, link = nil)
224      nombre1 = "UNIVERSIDAD CENTRAL DE VENEZUELA"
225      nombre2 = "FACULTAD DE CIENCIAS"
226      nombre3 = "COORDINACION ACADEMICA"
227      nombre4 = "DIVISION DE CONTROL DE ESTUDIOS"
228      ss = 10
229      pdf.add_text_wrap(pdf.margin_x_middle - (pdf.text_width(nombre1, ss) / 2.0),
230                        (pdf.absolute_top_margin-pdf.font_height(@font_size)), pdf.text_width(nombre1, ss), nombre1, ss)
231      pdf.add_text_wrap(pdf.margin_x_middle - (pdf.text_width(nombre2, ss) / 2.0),
232                       (pdf.absolute_top_margin-(2*pdf.font_height(@font_size))), pdf.text_width(nombre2, ss), nombre2, ss)
233      pdf.add_text_wrap(pdf.margin_x_middle - (pdf.text_width(nombre3, ss) / 2.0),
234                       (pdf.absolute_top_margin-(3*pdf.font_height(@font_size))), pdf.text_width(nombre3, ss), nombre3, ss)
235      pdf.add_text_wrap(pdf.margin_x_middle - (pdf.text_width(nombre4, ss) / 2.0),
236                        (pdf.absolute_top_margin-(4*pdf.font_height(@font_size))), 250, nombre4, ss)
237      pdf.add_image_from_file("public/images/logo_dce.jpg", (pdf.absolute_right_margin-50), (pdf.absolute_top_margin-50),
238                               width = 50, height = 50, link = nil)
239      pdf.text "\n", :font_size => 40, :justification => :center
240      pdf.text "\n", :font_size => 40, :justification => :center
241      pdf.select_font "Times-Bold"
242      pdf.text nombre, :font_size => 12, :justification => :center
```

Figura Nº 5-33. Código del primer fragmento del método exportar_pdf.

En este código lo primero que se hace es obtener los nombres de los campos a mostrar en el reporte (líneas 214-216). Luego se crea un documento PDF con hoja de tamaño carta y orientación vertical, en la línea siguiente se le asigna valor a los márgenes y se elige Times New Roman como tipo de letra. En la línea 220 se emplea el método start_page_numbering al cual hay que indicarle las coordenadas x,y donde debe ubicar la numeración de las páginas, el tamaño de letra y el patrón a utilizar para la numeración.

Luego se escribe el membrete del reporte, agregando inicialmente el logo de la universidad (línea 222), escribiendo luego el encabezado (líneas 229-235) y añadiendo posteriormente el logo de la División de Control de Estudios (línea 237), todos en una posición y tamaños específicos, y en la línea 242 se escribe el título del reporte.

Posteriormente se invoca al método max_columns (figura 5-34) el cual va a retornar un OHash con los anchos máximos de cada columna.

```
610  def max_columns(table1, columnas)
611      #Método que encuentra el ancho máximo de cada columna basandose en los datos contenidos en
612      #table1 y cabeceras contenidas en columnas
613        max_width = PDF::Writer::OHash.new(-1)
614        pdf = PDF::Writer.new
615          table1.data.each do |row|
616              row.each do |name, column|
617                  tamano = column.to_s.size
618                  if(tamano <= 20)
619                      w = pdf.text_width(column, @font_size)
620                  else
621                      temp = column[0..19]
622                      w = pdf.text_width(temp, @font_size) # ancho de los 20 primeros caracteres
623                  end
624                   w *= PDF::SimpleTable::WIDTH_FACTOR
625                   max_width[name] = w if w > max_width[name]
626              end
627          end
628        columnas.each do |name|
629          tamano = name.size
630          lineas = 1
631          if(tamano <= 20)
632              w = pdf.text_width(name, @font_size)
633          else
634              temp = name[0..19]
635              w = pdf.text_width(temp, @font_size) # ancho de los 20 primeros caracteres
636          end
637             w *= PDF::SimpleTable::WIDTH_FACTOR
638             max_width[name] = w if w > max_width[name]
639        end
640        return max_width
641  end
```

Figura Nº 5-34. Código del método max_columns.

Básicamente lo que hace este método es iterar sobre el resultado (líneas 615-627) y evaluar el ancho que ocupa cada valor, si este ocupa más de veinte caracteres entonces en el OHash se almacena que para esa columna el ancho máximo es el de los primeros veinte caracteres de ese valor, de lo contrario se almacena el ancho que ocupe dicho valor. Luego se obtienen los nombres de los campos a mostrar (líneas 628-639) y se aplica el mismo proceso de evaluación del ancho de cada valor.

El OHash retornado por este método es recibido en el método exportar_pdf y en base a los valores retornados se determina cuantas columnas van a caber por página. En caso de que todas las columnas de un reporte no quepan en una misma página, se hace toda la lógica de forma tal que al colocar las páginas una al lado de la otra, se puedan visualizar los resultados del reporte. En la figura 5-35 se muestra la porción del código de este método en la que se llena un hash por cada fila del resultado que se recorra (líneas 438-442), con los valores de todos los campos de esa fila. La variable table2 es un objeto PDF::SimpleTable el cual va a ser llenado con los resultados que quepan en dicha página, la variable max indica hasta cual columna se debe escribir en esa página y la variable ii indica a partir de cual columna se debe escribir en esa página. Una vez finalizado el recorrido de una fila del resultado, el hash es agregado a los datos del objeto SimpleTable (línea 443), y así sucesivamente se van recorriendo todas las filas del resultado que quepan en dicha página. Se puede observar la llamada al método convert_encoding en la línea 439, esto debido a

que la librería PDF::WRITTER no tiene compatibilidad con la codificación UTF8 y por lo tanto resulta necesario hacer la conversión a la codificación latina ISO8859-1, para que las letras acentuadas o con tildes sean escritas correctamente.

```
437         aux = Hash.new
438         while (ii < max)
439             valor = convert_encoding(row[ii].to_s)
440             aux.merge!({columnas[ii] => valor})
441             ii = ii+1
442         end
443         table2.data << aux
```

Figura Nº 5-35. Código del segundo fragmento del método exportar_pdf.

Una vez finalizado esto, en la figura 5-36 se muestra la forma como se obtienen los nombres de la cabecera de la tabla a mostrar en la página (línea 457) y se le asigna al objeto SimpleTable (línea 466). También se verifica si el usuario escogió la plantilla de formato azul y se asignan los tonos de azul a la cabecera y a las filas de la tabla (líneas 459-465) y este objeto SimpleTable es escrito en el PDF.

```
456         orden = Array.new
457         orden = columnas[ii..(max-1)]
458         #Si el usuario escogió el formato azul
459         if (formato_pdf=="Formato azul")
460             azul = Color::RGB.new(0x87, 0xce, 0xeb).freeze
461             azulclaro = Color::RGB.new(0xad, 0xd8, 0xe6).freeze
462             table2.shade_headings = true
463             table2.shade_heading_color = azul
464             table2.shade_color = azulclaro
465         end
466         table2.column_order = orden
467         table2.maximum_width = pdf.margin_width
468         if ((ap_filas == 0) && (j > 0))
469             pdf.y = y
470         end
471         table2.render_on(pdf)
```

Figura Nº 5-36. Código del tercer fragmento del método exportar_pdf.

Luego se evalúa si aún hay filas sin escribir en el PDF, de ser así se inserta una nueva página y se repite el proceso de escribir las columnas que correspondan a dicha página.

En la figura 5-37 se muestra el fragmento de código que se corresponde al reemplazo de los espacios en blanco por el carácter guión bajo en el nombre del reporte (línea 514), en la siguiente línea se le concatena la fecha y hora actual como tipo de dato entero para identificar al

reporte de forma unívoca y luego se abre el archivo en modo de escritura y se invoca al método render del objeto PDF el cual se encarga de escribir el contenido del objeto.

```
514         nombre1 = nombre.gsub(' ', '_')
515         nombre1 = nombre1.dup << "_"<<(Time.now.to_i).to_s
516         if(File.open("#{RAILS_ROOT}/public/reporte/#{nombre1}.pdf", "wb") { |f| f.write pdf.render })
517             mensaje ='El reporte fue exportado exitosamente a formato PDF.'
518
519         else
520             mensaje ='El reporte no fue exportado.'
521         end
```

Figura Nº 5-37. Código del cuarto fragmento del método exportar_pdf.

5.7.5 Pruebas

Las pruebas de aceptación para la exportación de reportes a formatos PDF y Excel, se realizó verificando que los resultados mostrados en estos tipos de archivos eran los mismos que los arrojados por consultas SQL sobre la base de datos.

5.8 Iteración 7

5.8.1 Objetivo

En esta iteración se desarrollaron y se corrigieron algunos métodos que manejan los datos del reporte del lado del servidor, utilizando las tecnologías provistas por AJAX, obteniendo así un mejor desempeño de la aplicación.

5.8.2 Planificación

En la tabla 5-8 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
8.01	28/01/2009	Desarrollo de pasos 3 y 4 del wizard.	Corrección.

Tabla Nº 5-8: Historias de usuario. Iteración 7.

5.8.3 Diseño

Los atributos y métodos necesarios para elaborar los requerimientos son incorporados a la clase Reporte y se pueden observar en la figura 5-38.

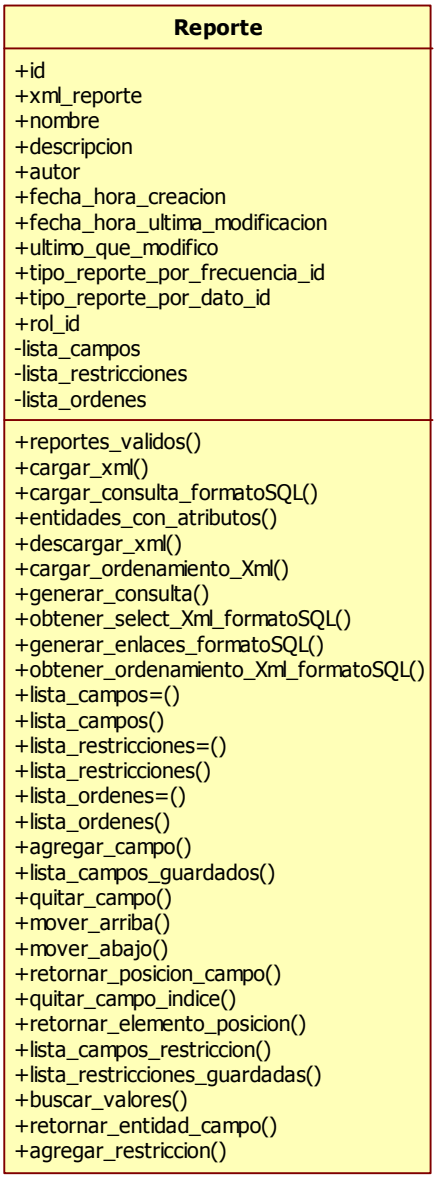


Figura Nº 5-38. Diagrama de la clase Reporte.

Se incorporaron tres nuevos atributos, estos son tres arreglos que se utilizarán para almacenar los campos, las restricciones y los campos de ordenamiento respectivamente. Es necesario crear estos arreglos para que en lado del servidor se vayan almacenando al momento los campos que el usuario seleccione para agregar, eliminar o mover. Anteriormente esto se hacía con funciones javascript y se almacenaban los cambios realizados del lado del servidor al ir de un paso a otro en el wizard y no en el mismo instante en el que el usuario efectuaba algún cambio de estos.

Los métodos cargar_campos_Xml y descargar_campos_Xml fueron eliminados ya que los campos están siendo guardados en el arreglo lista_campos y no directamente en el XML. Anteriormente los campos eran guardados directamente en el XML y esto desmejoraba el desempeño de la aplicación.

5.8.4 Codificación

En la figura 5-39 se puede observar el código correspondiente al método agregar_campo, el cual se encarga de agregar el campo seleccionado por el usuario a la lista de campos de la aplicación. Este método recibe tres argumentos: la entidad, el nombre y el alias del campo seleccionado por el usuario.

```
847  def agregar_campo (entidad_nuevo, campo_nuevo, campo_as_nuevo)
848    #Retorna la lista de campos con el nuevo campo o Retorna nil, si el campo ya está en la lista
849    #La lista temporal de campos del reporte es de la forma
850    #lista_campos=[{"entidad"=>entidad1,"campo"=>campo1,"campo_as"=>campo_as1},..,{"entidad"=>entidadN,"campo"=> campoN,"campo_as"=>campo_asN}]
851    lista_campos_actual = self.lista_campos
852    existe=false;
853    lista_campos_actual.each do |elemento|
854      if (elemento["entidad"].eql?(entidad_nuevo) && elemento["campo"].eql?(campo_nuevo))
855        # atributo duplicado
856        # no se puede agregar
857        existe=true;
858        break
859      end
860    end
861    if (existe)
862      # atributo duplicado, no se puede agregar
863      return nil
864    else
865      nuevo= {"entidad" => entidad_nuevo,"campo" => campo_nuevo,"campo_as" => campo_as_nuevo}
866      self.lista_campos= (lista_campos_actual.push(nuevo))
867      return self.lista_campos
868    end
869  end
```

Figura Nº 5-39. Código del método agregar_campo.

En la línea 851 se obtiene el arreglo de campos, se recorre cada uno de los campos que están guardados allí y se verifica si el elemento que llegó ya existe en el arreglo (línea 854), en caso de que no exista, en la línea 865 se crea un hash que contiene la entidad, el nombre y el alias del campo seleccionado y este hash es agregado al arreglo de campos del reporte.

A continuación (figura 5-40) se puede observar el código del método quitar_campo, el cual se encarga de eliminar un campo que haya seleccionado el usuario de la lista de campos de la aplicación. Este método recibe dos argumentos: el campo a eliminar y el arreglo de campos.

```
894   def quitar_campo (campo_nuevo,lista_campos_actual)
895     #Elimina el campo de la lista de campos y retorna la lista actual.Retorna nil, si el campo no se eliminó o no se encontró.
896     #La lista temporal de campos del reporte es de la forma
897     #lista_campos=[{"entidad"=>entidad1,"campo"=>campo1,"campo_as"=>campo_as1},..,{"entidad"=>entidadN,"campo"=>campoN,"campo_as"=>campo_asN}]
898
899     existe=false
900     counter=0
901     if (!lista_campos_actual.nil?)
902       lista_campos_actual.each do |elemento|
903         if (elemento["campo"].eql?(campo_nuevo))
904           #Se consiguió el elemento a eliminar
905           if (!(lista_campos_actual.slice!(counter).nil?))
906             existe=true
907             break
908           end
909         end
910         counter+=1
911       end
912     end
913     if (existe)
914       # atributo eliminado
915       return lista_campos_actual
916     else
917       # elemento no encontrado
918       return nil
919     end
920   end
```

Figura Nº 5-40. Código del método quitar_campo.

Lo primero que se hace es recorrer los campos del arreglo y una vez que se localiza en el arreglo el campo a quitar (línea 903), este es eliminado (línea 905) y finalmente se retorna el arreglo.

En la figura 5-41 se muestra el código del método mover_arriba, el cual es el encargado de implementar la funcionalidad de mover los campos seleccionados por el usuario una posición hacia arriba y recibe como parámetros un arreglo que contiene los campos a subir y el arreglo de campos del reporte.

En primera instancia, se recorre el arreglo de elementos a ser movidos y se invoca un método llamado retornar_posicion_campo el cual retorna un entero asociado a la posición que posee el elemento seleccionado en el arreglo de campos del reporte, es decir, un índice y en caso de no encontrarlo retorna un valor nulo.

Luego éstos valores enteros servirán para ubicar cada uno de los elementos seleccionados en el arreglo de campos del reporte, dado que los campos serán movidos en bloque hacia arriba, se valida que ninguno de los campos seleccionados se encuentre en la primera posición del arreglo, en caso de ser así se envía un mensaje de error al usuario (líneas 1050-1052).

En caso de que se puedan mover hacia arriba, se procede en base los índices de los campos seleccionados obtenidos anteriormente a remover cada uno de ellos del arreglo de campos del reporte con el método quitar_campo_indice (línea 1056), el cual remueve el elemento del arreglo y lo retorna, cada elemento que es removido es guardado y posteriormente dichos

elementos removidos son nuevamente insertados pero en una nueva posición en el arreglo de campos del reporte (línea 1065), en este caso su posición anterior disminuida en uno.

```
1034  def mover_arriba (campos_selected,lista_campos_actual)
1035    campos = campos_selected.to_a
1036    selLength = campos_selected.size
1037    posiciones = Array.new
1038    campos_seleccionados= Array.new
1039    selectedCount = 0
1040    error= false
1041    i=selLength-1
1042    while (i >=0)
1043      posicion = retornar_posicion_campo(campos[i],lista_campos_actual)
1044      if (!posicion.nil?)
1045        posiciones.push(posicion)
1046      end
1047      i=i-1
1048    end
1049    valor=""
1050    if (posiciones.include?(0))
1051      error=true
1052      valor= "Error: No se puede subir más."
1053    else
1054      posiciones.each { |pos|
1055        if (!pos.nil?)
1056          elemento= quitar_campo_indice(pos,lista_campos_actual)
1057          if(!elemento.nil?)
1058            campos_seleccionados.push(elemento)
1059            selectedCount = selectedCount+1
1060          end
1061        end
1062      }
1063      i=selectedCount-1
1064      while (i >=0)
1065        lista_campos_actual.insert(posiciones[i]-1, campos_seleccionados[i])
1066        i = i-1
1067      end
1068      error= false
1069      valor= "Se subieron los campos."
1070    end
1071    [valor,error]
1072  end
```

Figura Nº 5-41. Código del método mover_arriba.

De forma muy similar existe otro método que se encarga de mover los elementos hacia abajo, este método se llama mover_abajo y se muestra a continuación (figura 5-42). Este método recibe también como parámetro un arreglo que contiene los campos a ser movidos y recibe el arreglo en el que serán movidos.

```
985   def mover_abajo (campos_selected,lista_campos_actual)
986     campos = campos_selected.to_a
987     lista_campos_actual_size_antes=lista_campos_actual.size
988     selLength = campos_selected.size
989     posiciones = Array.new
990     campos_seleccionados= Array.new
991     selectedCount = 0
992     error= false
993     i=selLength-1
994     while (i >=0)
995       posicion = retornar_posicion_campo(campos[i],lista_campos_actual)
996       if (!posicion.nil?)
997         posiciones.push(posicion)
998       end
999       i=i-1
1000    end
1001    valor=""
1002    if (posiciones.include?((lista_campos_actual_size_antes-1)))
1003      error=true
1004      valor= "Error: No se puede bajar más."
1005    else
1006      posiciones.each { |pos|
1007        if (!pos.nil?)
1008          elemento= quitar_campo_indice(pos,lista_campos_actual)
1009          if(!elemento.nil?)
1010            campos_seleccionados.push(elemento)
1011            selectedCount = selectedCount+1
1012          end
1013        end
1014      }
1015      i=selectedCount-1
1016      while (i >=0)
1017        lista_campos_actual.insert(posiciones[i]+1, campos_seleccionados[i])
1018        i = i-1
1019      end
1020      error= false
1021      valor= "Se bajaron los campos."
1022    end
1023    [valor,error]
1024  end
```

Figura Nº 5-42. Código del método mover_abajo.

Siguiendo la misma idea del método mover_arriba, se recorre el arreglo de elementos a ser movidos y se guardan sus posiciones (índices) en el arreglo de campos del reporte, pero ahora como se moverán los campos hacia abajo, se valida que ninguno de los campos seleccionados se encuentre en la última posición del arreglo, en caso de ser así se envía un mensaje de error al usuario (líneas 1002-1004).

En caso de que se puedan mover hacia abajo, se procede en base a los índices de los campos seleccionados guardados anteriormente a remover cada uno de ellos del arreglo de campos del reporte (1008) y posteriormente dichos elementos removidos son nuevamente insertados pero en una nueva posición, en el arreglo de campos del reporte (línea 1017), en este caso su posición anterior incrementada en uno.

En la figura 5-43 se muestra un fragmento del código correspondiente al método agregar_restriccion, el cual se encarga de guardar los datos asociados a una restricción en la lista de restricciones del reporte. Básicamente lo que se hace en este método es verificar el tipo de restricción seleccionada por el usuario (igual, mayor que, menor que, etc.) y en base a esta se

pueden recibir uno o dos valores. Posteriormente en la línea 1521 se construye un hash en el que se guarda la entidad, el campo concatenado con un id que lo identifica unívocamente, el nombre del campo, el tipo de restricción, el string que se muestra en la lista de restricciones de la interfaz y el valor o valores. Luego este hash es agregado al arreglo de restricciones (línea 1523).

```
1521    nuevo= {"entidad"=> entidad_restriccion,"campo" => ((Time.now.to_i.to_s)<<campo_restriccion),"atributo" => campo_restriccion,
1522          "restriccion" => tipoRestriccion, "visual" => visual, "valor" => valores}
1523    lista_restricciones_actual.push(nuevo)
```

Figura Nº 5-43. Código del método agregar_restriccion.

5.8.5 Pruebas

Para probar que en los arreglos se estaban guardando los datos correctamente, se hizo que el código tuviera algunas líneas que generaran salida en consola y se obtuvieron los resultados esperados.

5.9 Iteración 8

5.9.1 Objetivo

En esta iteración se desarrollaron y se corrigieron algunos métodos que manejan el ordenamiento de los resultados del reporte del lado del servidor, utilizando las tecnologías provistas por AJAX, obteniendo así un mejor desempeño de la aplicación. También se desarrollaron las funcionalidades que permitieran realizar cálculos en los reportes de listado.

5.9.2 Planificación

En la tabla 5-9 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
9.01	11/02/2009	Desarrollo del paso 5 del wizard.	Corrección
12.00	11/02/2009	Elaboración de funcionalidades de cálculo.	Nueva

Tabla Nº 5-9: Historias de usuario. Iteración 8.

5.9.3 Diseño

A la clase Reporte se le incorporaron los atributos y métodos necesarios para solventar los requerimientos de esta iteración (Ver figura 5-44).

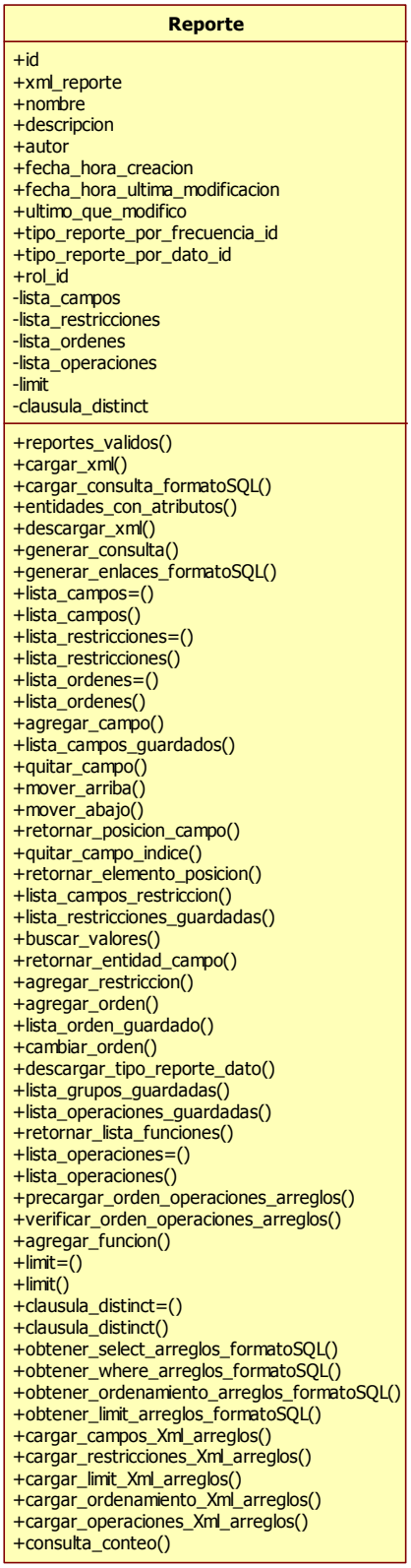


Figura Nº 5-44. Diagrama de la clase Reporte.

El atributo lista_operaciones es un arreglo en el que se van a almacenar las funciones definidas por el usuario sobre un campo, el atributo limit es un arreglo en el que se va a almacenar el límite inferior y el límite superior que acotarán los registros que se desean obtener y el atributo clausula_distinct es un string que está inicializado en “si” por defecto e indica la selección de registros distintos al construir la cláusula select de la consulta del reporte.

El método cargar_ordenamiento_Xml fue eliminado de la clase Reporte ya que los campos de ordenamiento están siendo almacenados en el arreglo de ordenes y no en el XML. Los métodos obtener_select_Xml_formatoSQL y obtener_ordenamiento_Xml_formatoSQL también fueron eliminados y en su lugar están siendo invocados los métodos obtener_select_arreglos_formatoSQL y obtener_ordenamiento_arreglos_formatoSQL, para construir la cláusula select y la cláusula de ordenamiento de la consulta del reporte respectivamente.

A continuación (figura 5-45) se muestra una nueva interfaz asociada al paso 6 del wizard, donde se solicita al usuario de manera opcional que indique si desea definir algún tipo de función sobre el campo seleccionado. Cabe destacar que en la lista de la derecha se muestran los campos de ordenamiento que el usuario definió en el paso anterior y a medida que el usuario va seleccionando campos de la lista de la izquierda, se muestran en la parte superior derecha unas listas para que el usuario indique a cual grupo desea asociar la función y la operación que desea realizar sobre ese campo. Si el campo es de tipo string sólo se ofrecen las funciones de contar y contar distintos, si el campo es numérico se ofrecen las funciones anteriores más las funciones de suma y promedio.

Figura Nº 5-45. Vista paso 6.

5.9.4 Codificación

En la figura 5-46 se puede observar el código correspondiente al método agregar_orden, el cual se encarga de agregar el campo seleccionado por el usuario a la lista de campos de ordenamiento del reporte. Este método recibe por parámetros el campo a agregar a la lista de ordenes y la entidad correspondiente a este campo.

```
1764    def agregar_orden (entidad_nuevo, campo_nuevo)
1765      #Retorna la lista de campos de ordenamiento con el nuevo campo o Retorna nil, si el campo no se agrego
1766      #La lista temporal de campos de ordenamiento del reporte es de la forma
1767      #lista_ordenes=[{"entidad"=>entidad1,"campo"=>campo1,"orden"=>"ASC"},..,{"entidad"=>entidadN,"campo"=>campoN,"orden"=>"ASC"]]
1768
1769      lista_orden_actual = self.lista_ordenes
1770      existe=false
1771      lista_orden_actual.each do |elemento|
1772        if (elemento["entidad"].eql?(entidad_nuevo) && elemento["campo"].eql?(campo_nuevo))
1773          # atributo duplicado
1774          # no se puede agregar
1775          existe=true
1776          break
1777        end
1778      end
1779
1780      if (existe)
1781        # atributo duplicado, no se puede agregar
1782        return nil
1783      else
1784        nuevo= {"entidad" => entidad_nuevo,"campo" => campo_nuevo,"orden" => "ASC"}
1785        self.lista_ordenes= (lista_orden_actual.push(nuevo))
1786        return self.lista_ordenes
1787      end
1788    end
```

Figura Nº 5-46. Código del método agregar_orden.

Lo primero que se hace es recorrer todos los campos que se encuentran en el arreglo de ordenes y se verifica si la entidad y el campo recibidos ya se encuentran en el arreglo (línea 1772). En caso de que no existan en el arreglo, se construye un hash (línea 1784) en el que se almacena la entidad, el campo y el orden, este último se guarda como ascendente por defecto. Este hash es agregado al arreglo de ordenes (línea 1785) y este último es retornado.

El método paso_6 de la clase Controlador ya no invoca a métodos para guardar los campos de ordenamiento en el XML del reporte, ahora sólo redirecciona a la nueva interfaz del paso 6. En esta interfaz ya se tiene cargada en la lista de la derecha los campos de ordenamiento agregados en el paso anterior. Estos son obtenidos a través del método precargar_orden_operaciones_arreglos, cuando el usuario está por primera vez en la interfaz del paso 6, o se pueden obtener a través del método verificar_orden_operaciones_arreglos de la clase Reporte (figura 5-47) cuando el usuario navega por la opción de Regresar.

En este método lo que se hace es obtener el arreglo de ordenes, se toma cada uno de los campos contenidos en él y se verifica si ese campo ya existe como un grupo en el arreglo de operaciones (línea 2874). En caso de que no exista, en la línea 2881 se construye un hash en el que se guarda la entidad, el campo y un arreglo con operaciones asociadas a este grupo y este es guardado en el arreglo de lista de operaciones (línea 2882). Posteriormente se verifica si existe algún grupo en la lista de operaciones que ya no exista en el arreglo de ordenes y de ser así en la línea 2898 se elimina dicho grupo.

```
2861  def verificar_orden_operaciones_arreglos (lista_operaciones_actual)
2862
2863     lista_ordenes_actual = self.lista_ordenes
2864     if (!lista_operaciones_actual.nil?)
2865       # lista_operaciones = [{"grupo_entidad" => entidad, "grupo_atributo" => campo, "operacion" =>{funciones}}]
2866       # funciones = [{"campo" => timestamp_campo1, "tipo_funcion"=> tipo,"atributo"=> campo,"entidad"=> entidad }]
2867
2868       #Precargo los ordenes que existen y verifico
2869       existe= false
2870       lista_ordenes_actual.each { |valor|
2871          existe = false
2872          grupo_atributo= valor["campo"]
2873          lista_operaciones_actual.each{|elemento|
2874             if (grupo_atributo.eql?(elemento["grupo_atributo"]))
2875               existe = true
2876               break
2877             end
2878          }
2879       if (!existe)
2880         operacion = Array.new
2881         nuevo= {"grupo_entidad"=> valor["entidad"],"grupo_atributo" => valor["campo"] ,"operacion" => operacion}
2882         lista_operaciones_actual.push(nuevo)
2883       end
2884       }
2885       #Elimino los ordenes que no existen y verifico
2886       existe= false
2887       index=0
2888       lista_operaciones_actual.each{|elemento|
2889          existe = false
2890          grupo_atributo= elemento["grupo_atributo"]
2891          lista_ordenes_actual.each { |valor|
2892             if (grupo_atributo.eql?(valor["campo"]))
2893               existe = true
2894               break
2895           end
2896          }
2897          if (!existe)
2898            lista_operaciones_actual.slice!(index)
2899          end
2900          index= index+1
2901       }
2902    end
2903     return lista_operaciones_actual
2904  end
```

Figura Nº 5-47.Código del método verificar_orden_operaciones_arreglos.

Cuando el usuario agrega un campo y selecciona a cual grupo lo va a asociar y que operación va a aplicarle, se invoca al método agregar_funcion cuyo código se muestra en la figura 5-48. Este método recibe por parámetros el grupo al que va asociada la función, la entidad del grupo, el campo sobre el cual se va a aplicar la operación, la entidad del campo y la función.

```
3023  def agregar_funcion (grupo,entidad_grupo,campo_operable,entidad_campo_operable,tipo_funcion)
3024
3025     lista_operaciones_actual = self.lista_operaciones
3026     lista_operaciones_actual.each{|elemento|
3027
3028        temp_entidad=elemento["grupo_entidad"]
3029        temp_atributo=elemento["grupo_atributo"]
3030
3031        if (entidad_grupo.eql?(temp_entidad)&& grupo.eql?(temp_atributo))
3032
3033           tag_function = {"campo" => ((Time.now.to_i.to_s)<< campo_operable), "tipo_funcion"=> tipo_funcion,
3034                           "atributo"=> campo_operable,"entidad"=> entidad_campo_operable}
3035           operacion= elemento["operacion"]
3036           operacion.push(tag_function)
3037        end
3038     }
3039     return lista_operaciones_actual
3040  end
```

Figura Nº 5-48.Código del método agregar_funcion.

En este método se obtiene el arreglo de operaciones y se busca el grupo al que se va a asociar la función en este arreglo. Una vez encontrado el grupo, en la línea 3033 se construye un hash con el nombre del campo concatenado a un número que lo va a identificar unívocamente, con la función, el campo y la entidad. Este hash es incorporado al arreglo “operacion” que se encuentra en el elemento actual (línea 3036) y finalmente la lista de operaciones actualizada es retornada.

Cuando el usuario presiona el botón continuar en la interfaz del paso 6, se invoca al método paso_7 de la clase Controlador, el cual se encarga de generar la consulta y la guarda en el XML. También se encarga de guardar en el XML los campos, las restricciones, el límite, el ordenamiento y los grupos con sus funciones asociadas y redirecciona a la interfaz elegir_formato, donde el usuario puede seleccionar el formato al que desea exportar el reporte.

En el método exportar_reporte de la clase Controlador se invoca a consulta_conteo de la clase Reporte (figura 5-49), el cual básicamente se encarga de recorrer las funciones asociadas a cada grupo e itera sobre el resultado del reporte, guardando los valores encontrados en un arreglo (línea 2156), hasta que cambia el valor del campo del grupo, de ser así se identifica cual es la función recibida y si no hay más funciones asociadas a ese grupo, pero hay otros grupos con funciones asociadas (línea 2162), se procede a invocar de forma recursiva al método (línea 2164). En caso de que hayan más funciones asociadas a ese grupo o no hayan ni más grupos ni más funciones, se construye un hash (línea 2170) en el que se almacena el campo en base al cual se agrupó, el valor del campo del grupo actual, el resultado obtenido para ese valor y un arreglo con los resultados obtenidos de las otras funciones y así sucesivamente se continúa iterando en el resultado hasta que finalmente se retorna un arreglo de hash que tienen la forma indicada anteriormente.

```
2155                if (valor_order == valor_anterior) && (limite_superior < ls)
2156                    valores << row[contador_campo]
2157                else
2158                    if (limite_superior >= ls)
2159                        break
2160                    else
2161                        if (funcion.eql?("conteo"))
2162                            if (tamano == 0) && ((indice+1) < lista_operaciones_actual.size)
2163                                hijo = Array.new
2164                                hijo = consulta_conteo(results,(indice+1),limite_inferior,limite_superior)
2165                                if (!hijo.nil?)
2166                                    nuevo = {"order" => order, "valor_order" => valor_anterior,
2167                                             "resultado" => valores.size, "hijo" => hijo}
2168                                end
2169                            else
2170                                nuevo = {"order" => order, "valor_order" => valor_anterior,
2171                                         "resultado" => valores.size, "hijo" => nil}
2172                            end
2173                            todos << nuevo
2174                        elsif (funcion.eql?("suma"))
2175                            #Convertimos los valores a float
```

Figura Nº 5-49.Código de un fragmento del método consulta_conteo.

5.9.5 Pruebas

Para probar que en los arreglos y en el XML se estaban guardando los datos correctamente, se hizo que el código tuviera algunas líneas que generaran salida en consola y se obtuvieron los resultados esperados.

Las pruebas de aceptación de las funciones de cálculo consistieron en verificar que los resultados eran los mismos que los arrojados por consultas SQL sobre la base de datos.

5.10 Iteración 9

5.10.1 Objetivo

Se desarrollaron las funcionalidades que permitieran mostrar los resultados de los cálculos realizados en forma de tabla, para todos los formatos soportados por el sistema (CSV, Excel y PDF). También se elaboró la interfaz donde el usuario indica en base a que campos desea agrupar por filas y por columnas, así como el campo que desea calcular.

5.10.2 Planificación

En la tabla 5-10 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
13.00	25/02/2009	Elaboración de reporte de tipo de dato Conteo.	Nueva
9.02	25/02/2009	Desarrollo del paso 5 del wizard.	Mejora

Tabla Nº 5-10: Historias de usuario. Iteración 9.

5.10.3 Diseño

Para dar solución a los requerimientos de esta iteración fue necesario incorporar un nuevo atributo y varios métodos a la clase Reporte (figura 5-50).

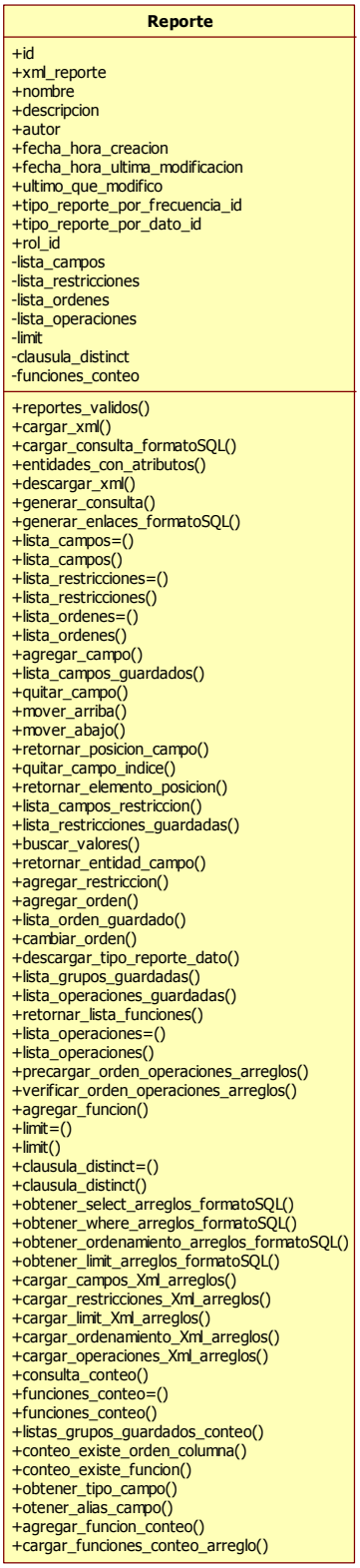


Figura Nº 5-50. Diagrama de la clase Reporte.

El atributo funciones_conteo es un arreglo que contiene el campo al que se le va a aplicar la función, la función a aplicar y la entidad a la que pertenece dicho campo.

A la clase Exportador también se le incorporaron nuevos métodos para imprimir el reporte de Tabla Cruzada en los diferentes formatos soportados por la aplicación (figura 5-51).

Figura Nº 5-51. Diagrama de la clase Exportador.

En la figura 5-52 se muestra la interfaz asociada a la selección de campos del reporte de tipo Tabla Cruzada.

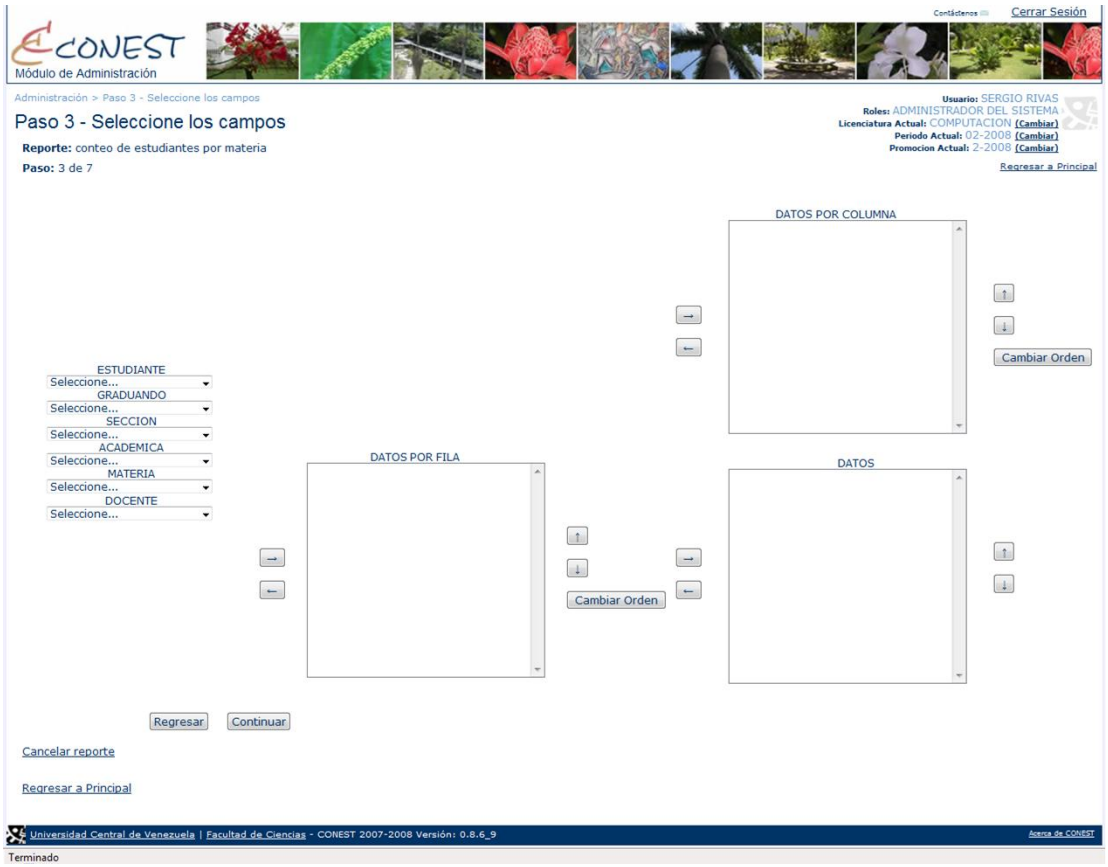


Figura Nº 5-52. Vista paso 3 Tabla Cruzada.

En esta interfaz se solicita al usuario que ingrese uno o más campos por filas, uno por columna y uno en la parte de datos, este último es al que se le va a aplicar la función.

En la figura 5-53 se puede apreciar la nueva interfaz del paso 5 de la opción Listado del wizard, en la que ahora se maneja toda la lógica correspondiente al ordenamiento y definición de funciones sobre los campos seleccionados. Este cambio fue necesario realizarlo debido a que se prestaba a confusión manejar en pasos separados el ordenamiento del resultado del reporte y la definición de funciones sobre estos ordenes, por lo tanto se decidió unir ambas funcionalidades en una sola interfaz.

Figura Nº 5-53. Vista paso 5 Definición de orden y funciones.

5.10.4 Codificación

Cuando el usuario selecciona algún campo de las listas de la parte izquierda de la interfaz opcion_conteo, se invoca al método seleccion_conteo de la clase Controlador y se verifica si el campo accion tiene el valor “agregar_columna” o “agregar_fila”, de ser así el campo es agregado al arreglo de campos y al arreglo de ordenes de la clase Reporte. Si el usuario va a eliminar algún campo, se invoca al método quitar_campo, el cual ya fue explicado en iteraciones anteriores.

Si el usuario agrega un campo para realizar algún cálculo sobre él, se verifica si ya existe un campo que tenga una función asociada, de no ser así se muestra al usuario una lista para que seleccione la función que desea aplicar, si el campo es de tipo texto las funciones provistas son contar y contar distintos, si es un campo numérico las funciones provistas son las anteriores más la función de suma. Una vez que el usuario indique la función y presione el botón Aceptar, se invoca al método agregar_funcion_conteo (figura 5-54) en el cual se construye un hash en el que se guarda el nombre del campo concatenado con un número que lo identifica unívocamente, la función, el nombre del campo, su entidad y el nombre del alias.

```
2727    def agregar_funcion_conteo (campo_operable,entidad_campo_operable,tipo_funcion, campo_operable_as)
2728
2729      lista_funciones_conteo = self.funciones_conteo
2730      # lista_operaciones = [{"grupo_entidad" => entidad, "grupo_atributo" => campo, "operacion" =>[funciones]}]
2731      # funciones = [{"campo" => timestamp_campo1, "tipo_funcion"=> tipo,"atributo"=> campo,"entidad"=> entidad ]}
2732
2733      tag_function = {"campo" => ((Time.now.to_i.to_s)<< campo_operable), "tipo_funcion"=> tipo_funcion,
2734              "atributo"=> campo_operable,"entidad"=> entidad_campo_operable, "atributo_as" => campo_operable_as}
2735      lista_funciones_conteo.push(tag_function)
2736
2737    return lista_funciones_conteo
2738
2739    end
```

Figura Nº 5-54. Código del método agregar_funcion_conteo.

Cuando el usuario oprime el botón Continuar se invoca al método paso_4, el cual redirecciona al usuario a la misma interfaz de restricciones de la opción Listado, sólo que cuando en esta interfaz del paso 4 el usuario oprime el botón Continuar, se invoca al método paso_6 y no a paso_5 como lo hace la opción de Listado y en este método es donde se procede a guardar en el XML los campos, las restricciones, los ordenes y funciones asociados al reporte y posteriormente se redirecciona a la vista elegir_formato para que el usuario seleccione el formato al que desea exportar el reporte.

Si el usuario desea exportar el reporte, se invoca al método consulta_conteo, el cual retorna un arreglo con los campos, sus funciones asociadas y los totales calculados. Luego se invoca al método grupos_fila_columna_conteo de la clase Reporte, el cual retorna el arreglo de campos por filas y el arreglo con el campo por columna. Posteriormente se procede a invocar el método de la clase Exportador que se corresponda, según el formato seleccionado por el usuario.

En la figura 5-55 se muestra el código del método imprimir_conteo_csv, el cual recibe por parámetros el nombre del reporte, el arreglo de funciones, el arreglo con el campo por columna y el arreglo con los campo por filas.

```
76  def imprimir_conteo_csv(nombre, global, order_columnas, order_filas)
77  nombre1 = nombre.gsub(' ', '_')
78  nombre1 = nombre1.dup << "_"<<(Time.now.to_i).to_s
79  FasterCSV.open("#{RAILS_ROOT}/public/reporte/#{nombre1}.csv", "w", {:force_quotes => true, :col_sep => ";"}) do |csv|
80      results = ""
81      # Imprimimos la cabecera del cross-tab
82      if(order_columnas.size > 0)
83      order_columnas.each do |posicion|
84      consulta = "select distinct " << posicion["order"] << " from " << posicion["entidad"] << " order by " << posicion["order"]
85      mysql = ActiveRecord::Base.connection
86      results = mysql.execute(consulta.dup << ";")
87      valores = Array.new
88      i = 0
89      while (i < order_filas.size)
90          if(i == 0) #Escribimos en la 1° celda los grupos que vinieron por fila y por columna
91              z = 0
92              celda1 = ""
93              order_filas.each do |elemento|
94                  if(z == 0)
95                      celda1 = celda1 << ("#{elemento["order"]} ")
96                      z += 1
97                  else
98                      celda1 = celda1 << ("--> #{elemento["order"]} ")
99                  end
100             end
101             celda1 = celda1 << ("/ #{posicion["order"]}")
102             valores << celda1
103         else
104             valores << ""
105         end
106         i += 1
107     end
108     results.data_seek(0)
109     results.each do |row|
110         valor = convert_encoding(row.to_s)
111         if(valor.eql?(""))
112             valores << "Sin valor"
113         else
114             valores << valor
115         end
116     end
117     valores << "Total"
118     csv << valores
119     end
120     else
121         results = nil
122     end
123     arreglo = Array.new
124     arreglo,booleano = imprimir_valores_conteo(global,csv,order_columnas,false,arreglo,results,true,true,0)
125     csv << arreglo
126 end
127 mensaje = 'El reporte fue exportado exitosamente a formato CSV.'
128 direccion = "#{nombre1}.csv"
129 [mensaje, direccion]
130
131 end
```

Figura Nº 5-55. Código del método imprimir_conteo_csv.

Al igual que en el método exportar_csv, se reemplazan los espacios en blanco por el carácter guión bajo (_) en el nombre del reporte y se le concatena un número que se obtiene a partir de convertir la fecha y hora actual como tipo de dato entero y el archivo se abre en la ruta indicada en modo escritura.

En la línea 84 se obtiene el campo que vino por columna y con él se construye una consulta en lenguaje SQL que arroja todos los posibles valores que puede tomar ese campo (línea 87). Luego se recorren los atributos que vinieron por filas, se toman sus nombres y se van concatenando para mostrarlos en la primera celda de la cabecera de la tabla, para indicar cuáles son los campos implicados en el reporte (líneas 94-101) y esto es agregado al arreglo valores.

Posteriormente los resultados arrojados por la consulta anterior son tomados y también se guardan en el arreglo valores (línea 115). En este arreglo también se guarda el string “Total” y se procede a escribir el contenido del arreglo en el archivo CSV (línea 119). En la línea 125 se invoca al método imprimir_valores_conteo el cual se encarga de escribir el contenido de la tabla a través de llamadas recursivas a sí mismo.

5.10.5 Pruebas

Para probar que en los arreglos y en el XML se estaban guardando los datos correctamente, se hizo que el código tuviera algunas líneas que generaran salida en consola y se obtuvieron los resultados esperados.

Las pruebas de aceptación de los cálculos asociados a los reportes de tipo Tabla Cruzada consistieron en verificar que los resultados de los reportes eran los mismos que los arrojados por consultas SQL sobre la base de datos.

5.11 Iteración 10

5.11.1 Objetivo

En esta iteración se mejoró el modelo de datos del sistema agregando una entidad nueva: Estudiante en Período. También se realizaron mejoras a toda la interfaz del sistema.

5.11.2 Planificación

En la tabla 5-11 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
1.04	11/03/2009	Modelo de datos para el repositorio de datos del sistema.	Mejora
14.00	11/03/2009	Interfaz del sistema en general.	Nueva

Tabla Nº 5-11: Historias de usuario. Iteración 10.

5.11.3 Diseño

Al modelo de datos se le incorporó la entidad Estudiante en Período, la cual se relaciona con la entidad Estudiante y va a almacenar por período todos los datos académicos del estudiante. En la figura 5-56 se muestra el nuevo modelo de datos del sistema.

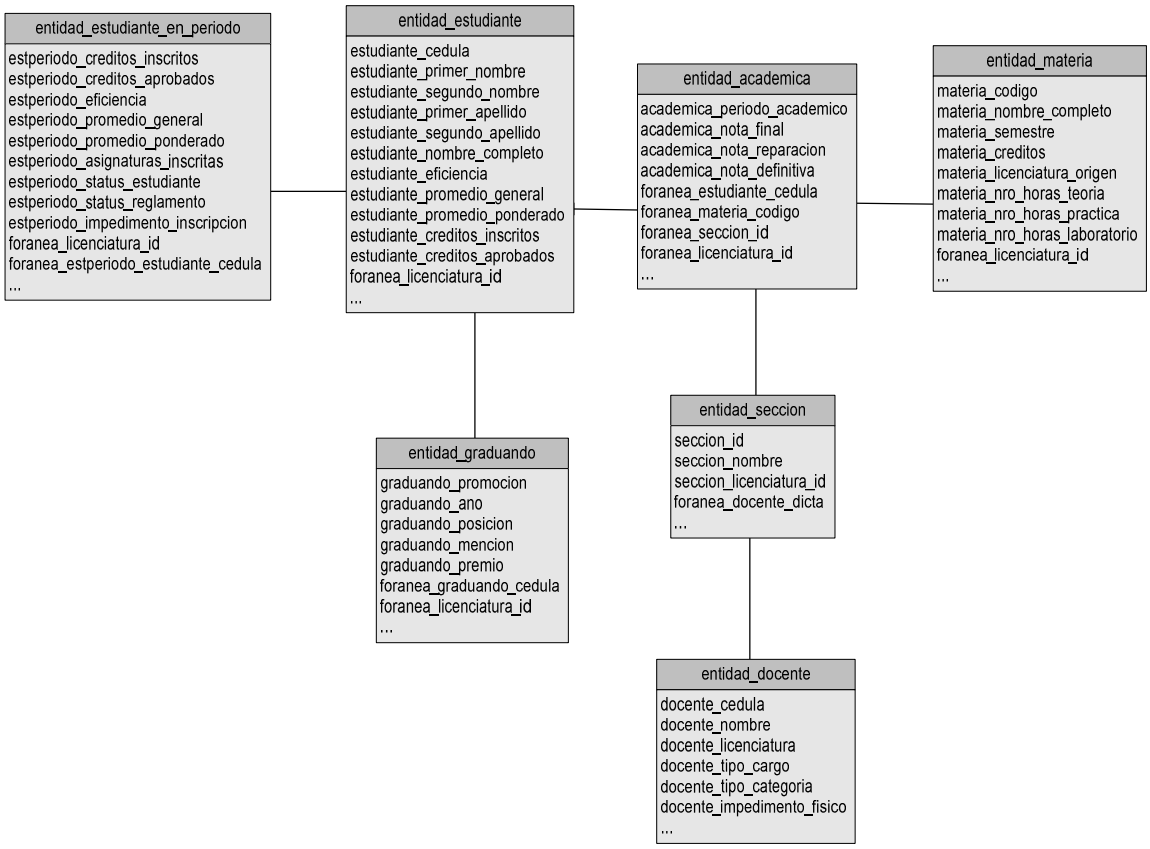


Figura Nº 5-56. Diagrama con nueva vista incorporada al repositorio de datos.

Por otro lado, fue mejorada la apariencia de las interfaces incorporándoles íconos y metáforas. En las figuras 5-57, 5-58 y 5-59 se muestran las nuevas interfaces de selección de campos en el reporte de tipo Listado, de definición de restricciones y de selección de campos en el reporte de tipo Tabla Cruzada respectivamente.

Figura Nº 5-57. Nueva vista paso 3.

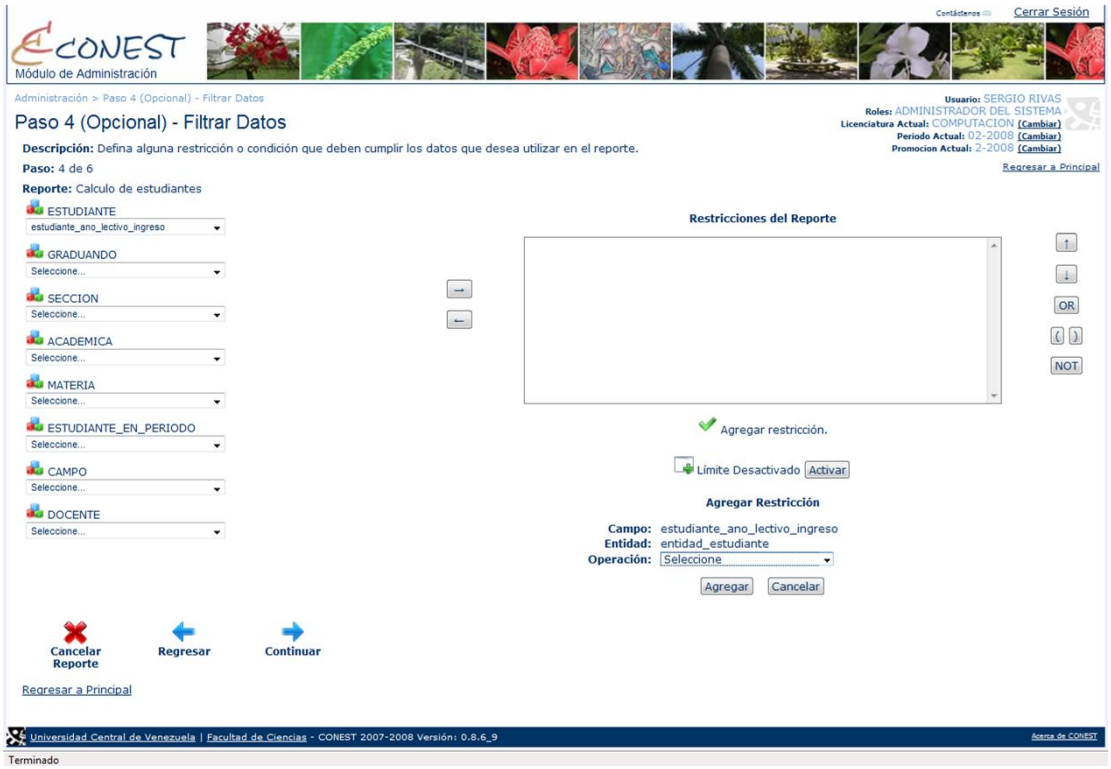


Figura Nº 5-58. Nueva vista paso 4.

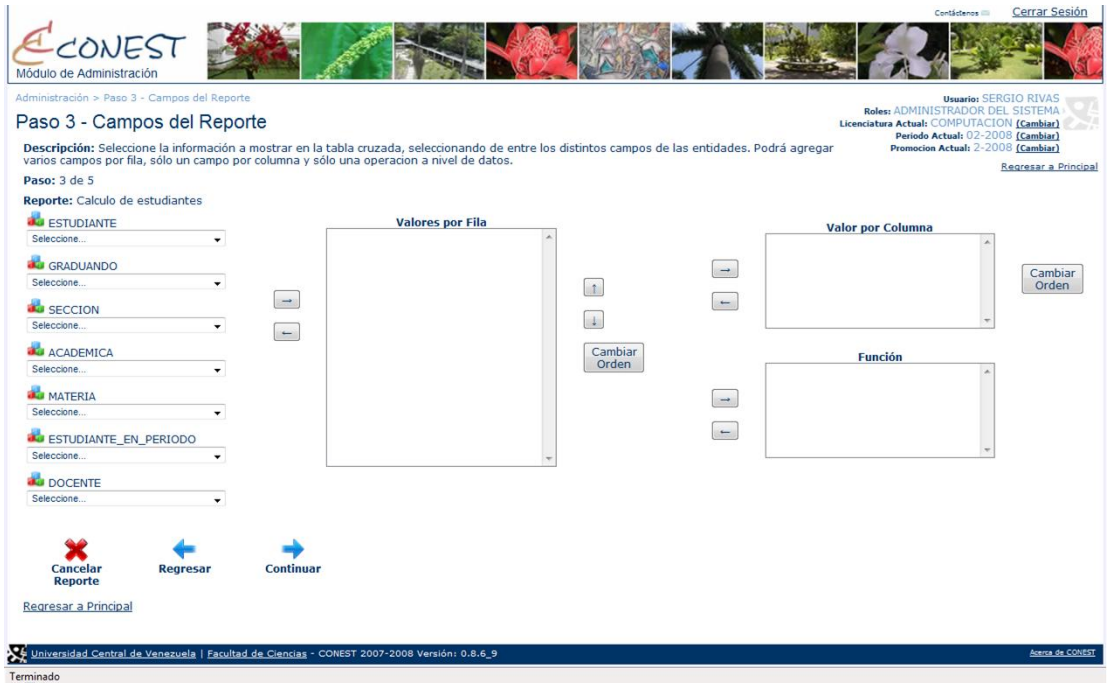


Figura Nº 5-59. Nueva vista paso 3 Tabla Cruzada.

En comparación con las interfaces anteriores que poseían estos pasos, ahora se hace un mejor aprovechamiento del espacio porque las listas de la parte izquierda se muestran cerradas y ya no están ocupando tanto espacio. También se tiene que los botones de Regresar y Continuar, así como la opción de Cancelar Reporte ahora son más llamativos por los íconos que tienen asociados.

5.11.4 Pruebas

Las pruebas de aceptación consistieron en mostrar a los usuarios las nuevas interfaces. Se obtuvo como resultado que los botones de la parte derecha y las opciones de seleccionar elementos distintos (paso 3) y limitar el resultado (paso 4) confundían al usuario. En cuanto a la interfaz del listado de reportes, los usuarios recomendaron separar el buscador y la opción de Crear Reporte de la lista de reportes.

5.12 Iteración 11

5.12.1 Objetivo

En esta iteración se realizaron mejoras en general a toda la interfaz del sistema, tomando como base las recomendaciones hechas por los usuarios en la iteración anterior. También se mejoraron los métodos que exportan los reportes de listado y tabla cruzada a formato PDF para que sólo exporten aquellos reportes que tengan una cantidad limitada de columnas y en base al número de estas modifique la orientación del reporte (vertical u horizontal). De igual forma se desarrollaron funcionalidades que permitieran mostrar los resultados del reporte en una vista preliminar en HTML, debido a que actualmente para conocer los resultados del reporte es necesario exportarlo a alguno de los formatos provistos.

5.12.2 Planificación

En la tabla 5-12 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
14.01	25/03/2009	Interfaz del sistema en general.	Mejora
10.01	25/03/2009	Exportación de resultados a formato PDF.	Mejora
15.00	25/03/2009	Desarrollo de funcionalidades para mostrar los resultados del reporte en HTML.	Nueva

Tabla Nº 5-12: Historias de usuario. Iteración 11.

5.12.3 Diseño

La nueva interfaz del paso 3 de los reportes de tipo listado, se muestra en la figura 5-60. Se puede notar que en la parte derecha sólo están los botones para subir y bajar los campos, en la parte inferior se muestra la opción de editar el nombre del campo y más abajo se encuentra la opción Avanzado, que actualmente sólo contiene la opción de seleccionar elementos distintos en estado activado.

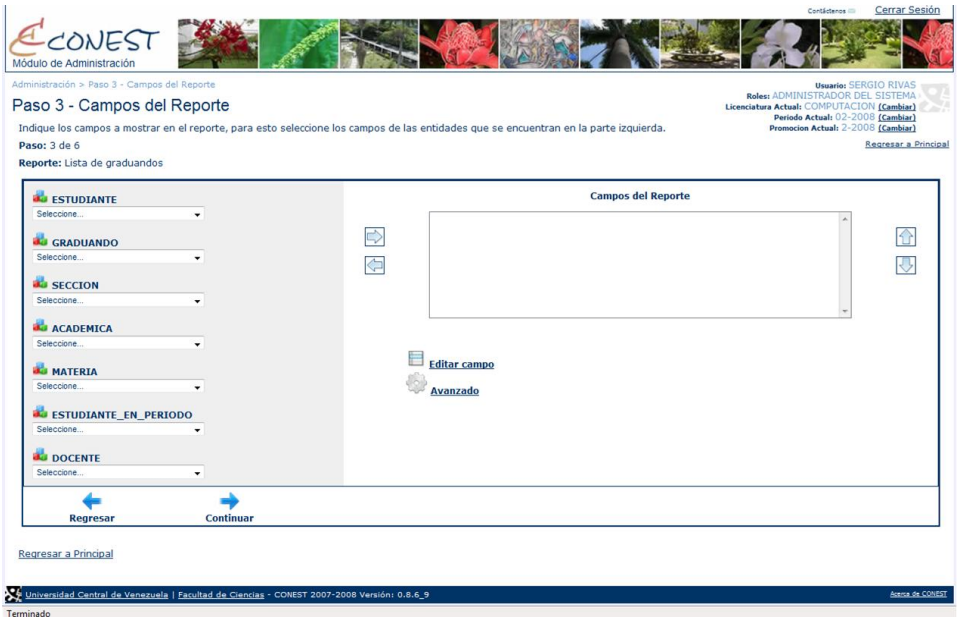


Figura Nº 5-60. Nueva mejora a la vista paso 3.

En la figura 5-61 se muestra la nueva interfaz del paso 4 del wizard. En la parte derecha ya no se muestran los conectores de las restricciones, actualmente estos son mostrados en la parte inferior y en la opción Avanzado se encuentra la limitación de la cantidad de registros del resultado.

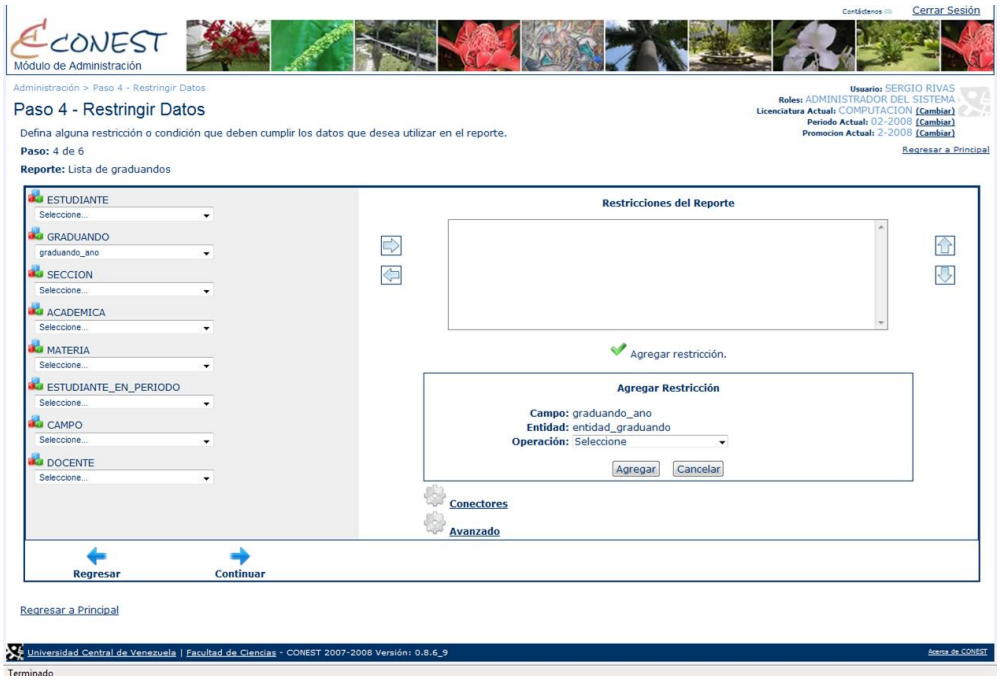


Figura Nº 5-61. Nueva mejora a la vista paso 4.

A continuación (figura 5-62) se muestra la nueva interfaz del paso 3 de los reportes de tipo Tabla cruzada. En la parte derecha de las listas de valores por fila y por columna ya no se muestra la opción de cambiar orden, ahora se muestra en la parte inferior de las listas.

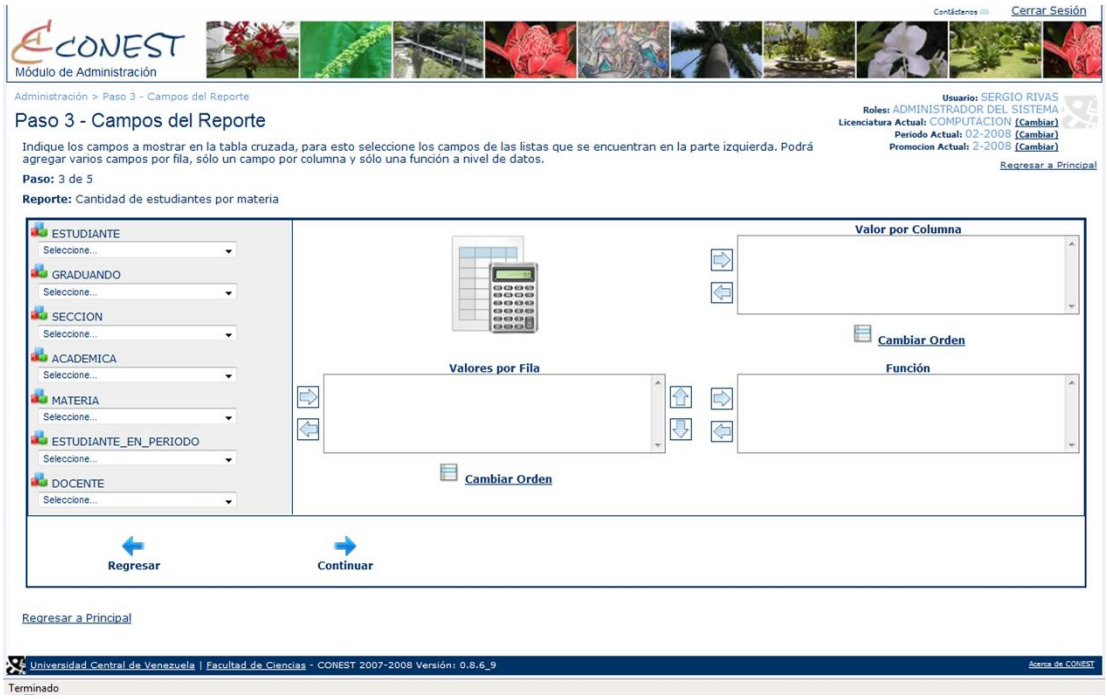


Figura Nº 5-62. Nueva mejora a la vista paso 3 Tabla Cruzada.

Un cambio que se aplicó a todas las interfaces es que ahora todos los componentes que forman parte del wizard se encuentran enmarcados dentro de un borde que los separa de los demás componentes de la interfaz, incluso la parte izquierda, donde se encuentran las listas con los campos de las entidades, posee un color de fondo distinto para separarla del resto de los componentes del wizard.

5.12.4 Codificación

En la figura 5-63 se muestra un fragmento del código del método exportar_pdf. En la línea 1295 se invoca al método max_columns el cual va a retornar un OHash con los anchos máximos de todas las columnas de la tabla, luego se obtiene el espacio disponible en la hoja con orientación vertical y horizontal (líneas 1298 y 1299 respectivamente).

```
1294 # Invocamos a un método que retorna el ancho máximo de cada columna de table1
1295 maximo = max_columns(table1, columnas, "lista")
1296
1297 gap = 2*table1.column_gap #gap de cada columna
1298 ancho_vertical = pdf.margin_width   #Ancho de la hoja vertical
1299 ancho_horizontal = pdf2.margin_width  #Ancho de la hoja horizontal
1300 ij = suma = contador = 0
1301 columnas_por_pagina = Array.new # Arreglo en el que se va a guardar la cantidad de columnas que caben en cada página
1302 ancho_por_pagina = Array.new # Arreglo en el que se va a guardar el ancho de cada tabla de cada página
1303 ancho_disp = ancho_vertical
1304 error = ""
1305 while (ij < columnas.length)
1306  if ((suma + maximo[columnas[ij]] + gap) <= ancho_disp)
1307      suma = suma + maximo[columnas[ij]] + gap
1308      contador = contador+1
1309  else
1310  if(ancho_disp < ancho_horizontal) #Todas las columnas del reporte no caben en la hoja con orientación vertical,
1311                                    #hay que probar si caben en la hoja con orientación horizontal
1312   ancho_disp = ancho_horizontal
1313   ij = -1
1314   suma = contador = 0
1315   columnas_por_pagina = Array.new #Arreglo en el que se va a guardar la cantidad de columnas que caben en cada página
1316   ancho_por_pagina = Array.new # Arreglo en el que se va a guardar el ancho de cada tabla de cada página
1317
1318  else #Las columnas del reporte no caben ni en orientación vertical ni horizontal
1319      error = 'El reporte contiene muchas columnas, por favor seleccione un formato distinto a PDF
1320               para exportar el reporte.'
1321      break
1322  end
1323  end
1324  ij = ij + 1
1325 end
1326 if(error != "")
1327     return error
1328 else
1329 columnas_por_pagina << contador
1330 ancho_por_pagina << suma
1331 gap2 = -5
1332 if (ancho_disp == ancho_horizontal)
1333     pdf = pdf2
1334     puts "Reporte Horizontal"
1335 end
1336 pdf.margins_cm(2, 3, 2, 2)
1337 pdf.select_font "Times-Roman"
1338 pdf.start_page_numbering(pdf.absolute_right_margin, (pdf.absolute_bottom_margin-(2*pdf.font_height(@font_size))),
1339                          9, nil, pattern = "<PAGENUM>/<TOTALPAGENUM>", 1)
1340 pdf.add_image_from_file("public/images/logo_ucv.jpg", pdf.absolute_left_margin, (pdf.absolute_top_margin-55),
1341                          width = 60, height = 60, link = nil)
```

Figura Nº 5-63. Fragmento del código del método exportar_pdf.

Posteriormente se obtienen los valores correspondientes al ancho de cada columna y se van sumando para determinar si caben en la hoja con orientación vertical (línea 1306), de no ser así se obtiene el espacio disponible en la hoja con orientación horizontal y se continúan sumando los valores de anchura de las columnas. Si el valor de esta suma es superior al del espacio disponible en la hoja horizontal, entonces se muestra un mensaje de error al usuario indicándole que seleccione otro formato distinto a PDF. En caso de que la sumatoria de los anchos de todas las columnas no supere el valor del espacio disponible, se obtiene el objeto PDF con la orientación indicada y se indican los valores de los márgenes y el tipo de letra a utilizar (líneas 1336 y 1337 respectivamente) y finalmente se procede a escribir los datos del reporte como se venía haciendo anteriormente.

Por otro lado, al método paso_6 de la clase Controlador se le incorporaron nuevas funcionalidades para que en la interfaz ahora se muestren los resultados del reporte, estos cambios básicamente consistieron en obtener el resultado del reporte y pasárselo a la vista correspondiente para que los muestre en la interfaz y de esta forma el usuario puede visualizarlos y comprobar su correctitud antes de exportarlos.

5.12.5 Pruebas

Las pruebas de aceptación para la exportación de reportes a formato PDF consistieron en generar una amplia variedad de reportes, donde aquellos que tenían poca cantidad de columnas fueron exportados sin ningún problema a formato PDF, mientras que con aquellos que poseían muchas columnas la aplicación emitía el mensaje esperado y no exportaba el reporte.

5.13 Iteración 12

5.13.1 Objetivo

En esta iteración se desarrollaron funcionalidades que permitieran manejar los reportes con tipo de frecuencia Regular de forma diferente a los de Por solicitud. Esto es porque los reportes Regulares poseen restricciones que son variables, como por ejemplo el año lectivo y el período académico, manteniéndose intactas las demás características del reporte. Por ello resulta necesario implementar métodos que manejen este tipo de restricciones variables, de forma tal que cuando el usuario emita un reporte de este tipo sólo se le solicite la información concerniente a estas restricciones y no ingrese nuevamente todos los datos del reporte.

5.13.2 Planificación

En la tabla 5-13 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
16.00	08/04/2009	Desarrollo de funcionalidades específicas para los reportes con tipo de frecuencia Regular.	Nueva

Tabla Nº 5-13: Historias de usuario. Iteración 12.

5.13.3 Diseño

En la figura 5-64 se muestra la interfaz de definición de restricciones en un reporte con tipo de frecuencia Regular.

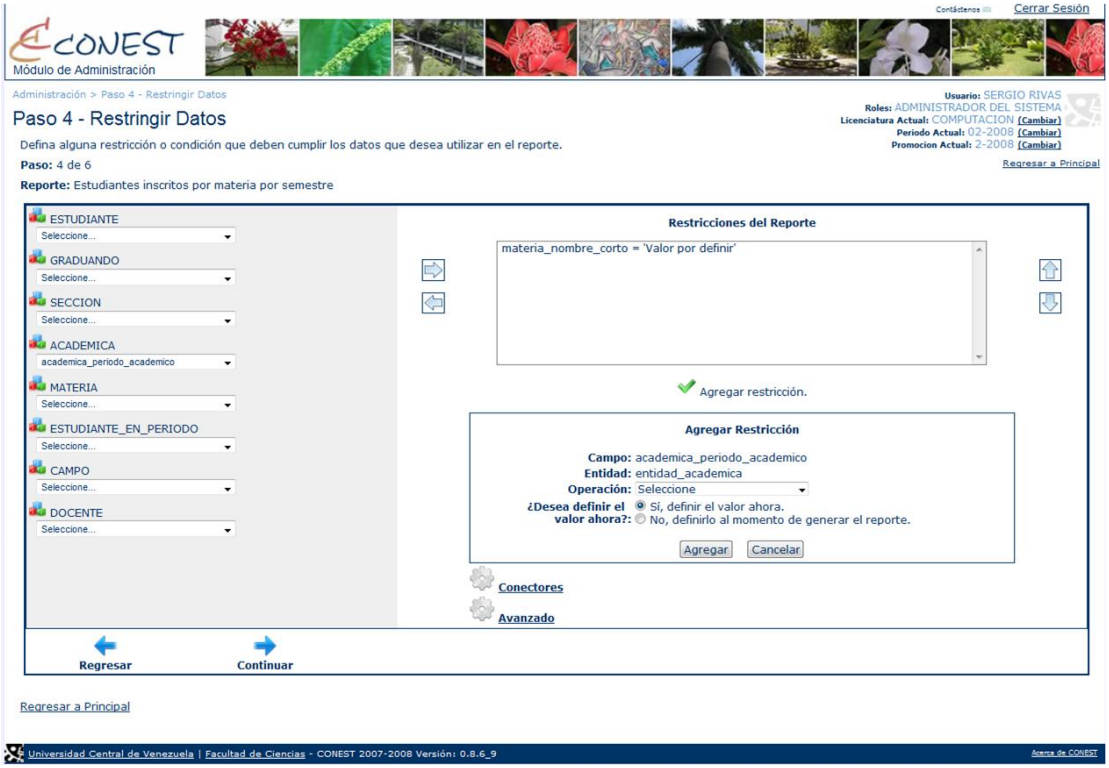


Figura Nº 5-64. Vista paso 4 de un reporte Regular.

Es de hacer notar que ahora se le solicita al usuario que indique si desea definir un valor para la restricción en ese momento o al generar el reporte, si elige la segunda opción se guardará esa restricción como de tipo variable y se solicitará su valor cada vez que el usuario genere el reporte.

En la figura 5-65 se muestra la interfaz del paso 6 de un reporte Regular donde se solicita al usuario que indique los valores de las restricciones variables.

Figura Nº 5-65. Vista paso 6 de un reporte Regular.

Una vez que el usuario ingrese los valores de las restricciones podrá generar el reporte y la aplicación le mostrará la interfaz con los formatos a los que puede ser exportado.

5.13.4 Codificación

En la figura 5-66 se muestra un fragmento del código del método agregar_restriccion, el cual ahora recibe un nuevo parámetro: tipo_valor. Si el valor de este parámetro es igual a “variable” (línea 1464), significa que el usuario seleccionó la opción de definir el valor al momento de generar el reporte, por lo tanto a los otros parámetros recibidos por el método, que deberían tener el valor o los valores de la restricción, se les asigna “Valor por definir” (líneas 1465-1468) y se coloca en “no” el valor de la variable opcion_completa (línea 1469). En caso contrario significa que el usuario seleccionó la opción de definir el valor en ese momento.

```
1461    valida =true
1462    opcion_completa = "si"
1463    if(!tipo_valor.nil?)
1464      if(tipo_valor.eql?("variable"))
1465        valor_tipo1 = "Valor por definir"
1466        input1Tipo3 = "Valor por definir"
1467        input2Tipo3 = "Valor por definir"
1468        input1Tipo2 = "Valor por definir"
1469        opcion_completa = "no"
1470      else
1471        tipo_valor="fijo";
1472      end
1473    else
1474      tipo_valor="fijo";
1475    end
```

Figura Nº 5-66. Fragmento del código del método agregar_restriccion.

En la figura 5-67 se muestra otra porción del código de este método donde se construye el hash con todos los datos de la restricción, solo que ahora se le incorpora el tipo de restricción y si está completa o no.

```
1546 if (valida)
1547  nuevo= {"entidad"=> entidad_restriccion,"campo" => ((Time.now.to_i.to_s)<<campo_restriccion),
1548            "atributo" => campo_restriccion, "restriccion" => tipoRestriccion, "visual" => visual,
1549            "valor" => valores, "tipoValor" => tipo_valor, "completa" => opcion_completa}
1550  lista_restricciones_actual.push(nuevo)
1551 end
1552 return lista_restricciones_actual
```

Figura Nº 5-67. Fragmento del código del método agregar_restriccion.

Al método paso_6 (figura 5-68) también se le incorporaron nuevas validaciones para verificar si todas las restricciones del reporte tienen valores asociados.

```
1053 def paso_6
1054   @indice = params['indice']
1055   previo_completo = params['previo_completo']
1056   reporte = buscar_reporte_sesion(@indice)
1057
1058     if(!reporte.nil?)
1059         @mostrar_reporte = true
1060         @roles= session[:usuario].todos_roles
1061         if(@roles.size == 1)
1062           for rol in @roles
1063             @rol = rol.id
1064           end
1065         end
1066 @campo_id,@entidad_variable,@campo_variable,@valores_variable,@tipo_variable,@tipo_restriccion=reporte.restriccion_variable_editar(reporte.lista_restricciones)
1067     if ((!previo_completo.nil?)&&(@campo_id.nil?))
1068       @ubicacion="wizard"
1069       @origen = reporte.generar_datos_reporte
1070       @restricciones_variables = reporte.obtener_restricciones_variables
1071     else
1072       @origen = reporte.generar_datos_reporte
1073       reporte.limpiar_restricciones_variables
1074       @ubicacion="wizard"
1075       @restricciones_variables = reporte.obtener_restricciones_variables
1076       guardar_reporte(@indice) #Invocamos el método que se encarga de guardar el reporte en el repositorio de datos
1077     end
1078
1079     if ((@restricciones_variables.nil?)||(!previo_completo.nil?))
1080         xmlDoc= Document.new reporte.xml_reporte
1081         consulta = xmlDoc.root.elements["consulta"].text
```

Figura Nº 5-68. Fragmento del código del método paso_6.

En la línea 1066 se invoca al método restriccion_variable_editar, el cual devuelve todos los datos concernientes a una restricción de tipo variable. Luego se valida si el usuario viene de la interfaz de inserción de valores en las restricciones variables y si definió los valores de todas estas (línea 1067), de ser así, se invoca al método generar_datos_reporte el cual guarda en el XML la consulta del reporte, los arreglos de campos, restricciones, ordenamiento, entre otros. De lo contrario, también se guarda en el XML todos los datos concernientes al reporte y se invoca al método limpiar_restricciones_variables el cual va a asignar “Valor por definir” a todas las restricciones de tipo variable y luego se invoca al método obtener_restricciones_variables el cual

retorna un arreglo con todas las restricciones de tipo variable y posteriormente se guarda el reporte en el repositorio de datos.

En la línea 1079 se verifica si no existen restricciones de tipo variable o si el usuario viene de la interfaz de inserción de valores en las restricciones variables, de ser así, se continúa con la ejecución del método paso_6 como se había explicado en iteraciones anteriores, de lo contrario se invoca al método restriccion_variable_editar, explicado anteriormente, y se redirecciona al usuario a la interfaz mostrada en la figura 5-65.

5.13.5 Pruebas

Para probar que en XML se estaban guardando los datos correctamente, se hizo que el código tuviera algunas líneas que generaran salida en consola y se obtuvieron los resultados esperados.

5.14 Iteración 13

5.14.1 Objetivo

En esta iteración se realizaron mejoras en general a toda la interfaz del sistema, se incorporó información de ayuda en cada paso del wizard para que el usuario conozca lo que puede realizar, se efectuó la integración de la aplicación con el sistema CONEST y también se realizaron pruebas y evaluaciones de usabilidad a la aplicación con el personal de la División de Control de Estudios encargado de realizar reportes.

5.14.2 Planificación

En la tabla 5-14 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
14.02	22/04/2009	Interfaz del sistema en general.	Mejora
17.00	22/04/2009	Incorporación de información de ayuda en cada paso del wizard.	Nueva
18.00	22/04/2009	Integración con el sistema CONEST.	Nueva
19.00	22/04/2009	Realizar evaluación de usabilidad del sistema.	Nueva

Tabla Nº 5-14: Historias de usuario. Iteración 13.

5.14.3 Diseño

En la figura 5-69 se muestra la nueva interfaz del paso 6 del wizard, ahora en esta interfaz la distribución de las acciones que se pueden realizar es más ordenada y no satura al usuario con tanta información como lo hacía anteriormente.

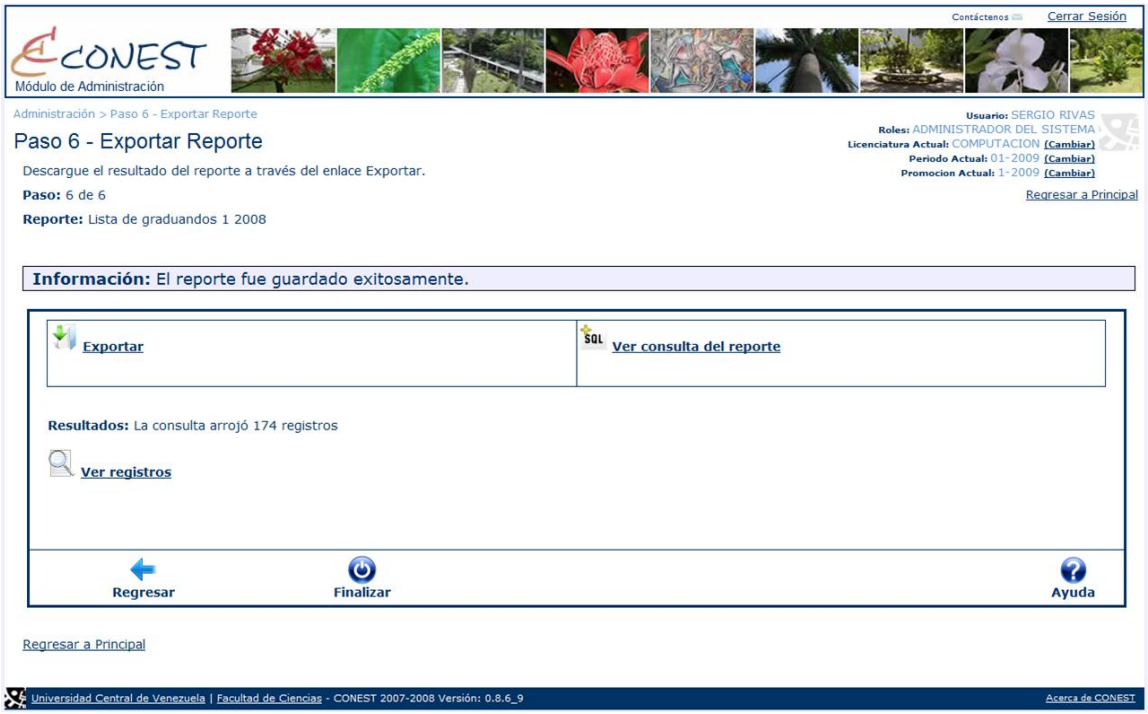


Figura Nº 5-69. Interfaz del paso 6 del wizard.

A continuación (figura 5-70) se muestra la interfaz de la ayuda que se brinda al usuario en el paso 3 del reporte de tipo listado. Al principio se muestra una imagen donde se señalan los diferentes sectores de ese paso del wizard y luego se ofrece una descripción de cada una de las acciones que el usuario puede realizar en ese paso.

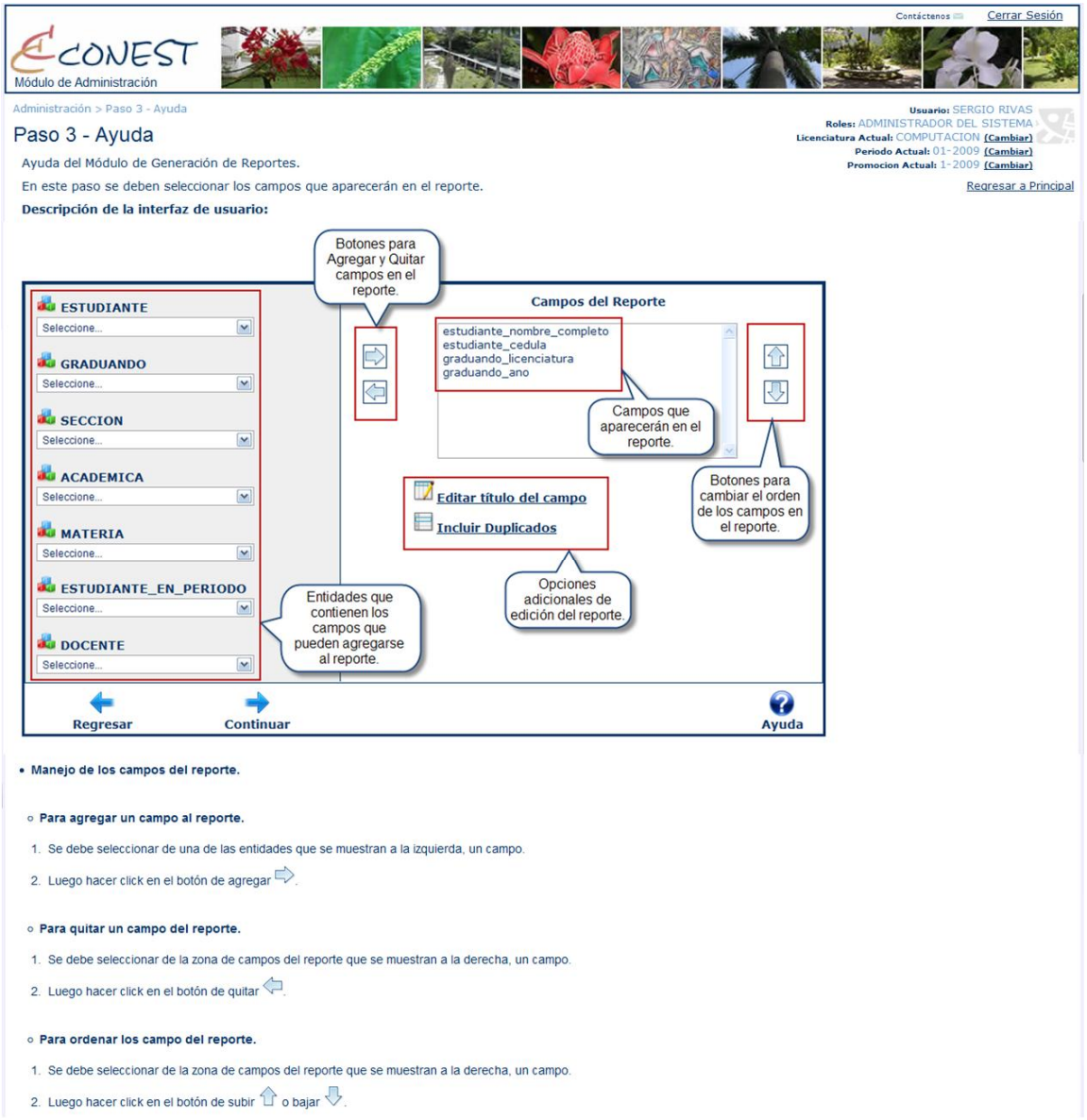


Figura N° 5-70. Ayuda del paso 3 del wizard.

También se diseñó el mecanismo de evaluación de usabilidad de la aplicación, el cual consiste en una encuesta de 20 preguntas con respuestas de selección simple, que buscan evaluar el funcionamiento de la aplicación y su interfaz. Las preguntas realizadas en la encuesta se pueden observar en el apéndice A del presente documento.

5.14.4 Codificación

En la figura 5-71 se muestra el código correspondiente al método “ayuda” de reporte_controller, el cual obtiene en la variable @ayuda a que paso del wizard corresponde la ayuda que debe mostrar y luego redirecciona a la vista del mismo nombre, mostrada en la figura 5-70.

```
2526 def ayuda
2527
2528  @ayuda = params['ayuda']
2529  @descripcion_paso = "Ayuda del Módulo de Generación de Reportes."
2530  @mostrar_reporte=true
2531
2532  if (@ayuda=="paso1")
2533    @titulo_pagina='Paso 1 - Ayuda'
2534  elsif (@ayuda=="paso2_sql")
2535    @titulo_pagina='Paso 2 - Ayuda'
2536  elsif (@ayuda=="paso2_wizard")
2537    @titulo_pagina='Paso 2 - Ayuda'
2538  elsif (@ayuda=="paso3_listado")
2539    @titulo_pagina='Paso 3 - Ayuda'
2540  elsif (@ayuda=="paso4")
2541    @titulo_pagina='Paso 4 - Ayuda'
2542  elsif (@ayuda=="paso5")
2543    @titulo_pagina='Paso 5 - Ayuda'
2544  elsif (@ayuda=="paso6_previo")
2545    @titulo_pagina='Paso 6 - Ayuda'
2546  elsif (@ayuda=="paso6_listado")
2547    @titulo_pagina='Paso 6 - Ayuda'
2548  elsif (@ayuda=="paso6_conteo")
2549    @titulo_pagina='Paso 6 - Ayuda'
2550  elsif (@ayuda=="paso3_conteo")
2551    @titulo_pagina='Paso 3 - Ayuda'
2552  elsif (true)
2553  end
2554
2555 end
```

Figura N° 5-71. Código del método ayuda.

Una vez que se consideró que el sistema estaba listo para ponerlo en producción, se realizó su implantación como parte de CONEST. Para ello fue necesario integrar al modelo de datos de CONEST los procedimientos almacenados y las vistas de la aplicación, así como también se incorporó el código fuente del generador de reportes. En general, no hubo complicaciones al momento de integrar con CONEST, debido a que la aplicación fue desarrollada tomando las medidas necesarias para no tener problemas de compatibilidad con las funcionalidades ya existentes en el sistema CONEST.

5.14.5 Pruebas

Las pruebas de usabilidad de la aplicación fueron realizadas a través de encuestas al personal de la División de Control de Estudios encargado de elaborar reportes, con la finalidad de obtener una retroalimentación por parte de los usuarios acerca de las funcionalidades y el aspecto de la interfaz del sistema. Las respuestas obtenidas fueron positivas, los usuarios se sentían satisfechos con el sistema. Se pudo observar que los usuarios presentaban dificultades al momento de elaborar los reportes de tipo Tabla Cruzada, debido a que no comprendían la estructura de este tipo de reporte, pero a medida que realizaban reportes de este tipo comprendían su utilidad.

También se hicieron pruebas a la aplicación, una vez que ya se encontraba integrada con CONEST, para evaluar su funcionamiento y los tiempos de respuesta. Estas pruebas consistieron en generar seis reportes: tres de tipo Listado y tres de tipo Tabla Cruzada. La especificación de cada una de estas pruebas se puede observar en el apéndice B del presente documento.

CONCLUSIONES

La investigación realizada para establecer el marco teórico, la adaptación de la metodología de programación extrema (XP) y el desarrollo de un software permitieron en este Trabajo Especial de Grado presentar como resultado una aplicación Web denominada CONEST Reportes, la cual es un módulo del sistema CONEST que permite la generación de reportes, que proveen a los usuarios la información contenida en el repositorio de datos de una manera concisa, en un formato de fácil lectura, logrando así cubrir los objetivos inicialmente propuestos.

Con la adaptación de la metodología XP, debido a la realización de pruebas a la aplicación y a la comunicación continua con el cliente, se logró obtener resultados cercanos a los pretendidos por el cliente.

Con esta aplicación, se provee al personal de la División de Control de Estudios de la Facultad de Ciencias de la UCV, una herramienta de fácil manipulación que no requiere de un entrenamiento exhaustivo para su utilización.

Para el diseño y construcción de esta aplicación se tomaron en cuenta aspectos de usabilidad, eficiencia, robustez y escalabilidad. En el desarrollo se utilizó el lenguaje de programación Ruby sobre el framework Rails, el manejador de bases de datos MySQL Server, que es el utilizado por el sistema CONEST, y un conjunto de librerías adicionales tales como PDF::WRITTER, Faster CSV y Ruby Spreadsheet que permitieron la exportación de los resultados del reporte a los formatos PDF, CSV y EXCEL respectivamente. Para el manejo de estas librerías fue necesario analizar su funcionamiento y realizar diversas pruebas que permitieran ir entendiendo progresivamente su funcionamiento. Para el equipo de desarrollo fue fácil aprender a utilizar estas librerías debido a la cantidad de documentación existente.

También se utilizó la librería REXML, caracterizada por tener un gran repertorio de funciones y métodos para el manejo de documentos XML, utilizados en esta aplicación para guardar información concerniente al reporte.

En el desarrollo de la aplicación Web, se planteó el diseño e implementación de una secuencia de pasos que asistiera al usuario en la elaboración del reporte. Esto se realizó en varias etapas, inicialmente la implementación de cada uno de los pasos involucró el desarrollo de funciones complejas en lenguaje Javascript, para manejar las distintas funcionalidades, las cuales eran ejecutadas en el navegador del cliente. De esta manera la comunicación entre el navegador Web que utilizara el usuario y el servidor Web donde se alojara la aplicación ocurría sólo al superar

un paso de la secuencia. Este enfoque probó no ser el más idóneo, ya que al ir avanzando en el desarrollo, incorporando cada vez más funcionalidades y al realizar las pruebas con el cliente, se comprobó que el desarrollo no era escalable y se complicaba cada vez más debido a problemas de compatibilidad con distintos navegadores y que la aplicación sufría de graves problemas de usabilidad.

Tras un análisis de la situación, se optó cambiar a un enfoque donde se utilizaría la tecnología AJAX y en el que el servidor Web jugaría un papel mayor, debido a que se decidió que las funcionalidades anteriormente implementadas en lenguaje Javascript y ejecutadas en el navegador, se implementarían en lenguaje Ruby y serían ejecutadas en el servidor. En este enfoque para el navegador sólo se desarrollarían las funciones necesarias para utilizar la tecnología AJAX y funciones que permitan una mejor interacción del usuario con la interfaz de usuario, como efectos gráficos, validaciones de datos, etc.

También es importante resaltar, que las interfaces gráficas de usuario (GUI) fueron desarrolladas tomando como base las plantillas del sistema CONEST y posteriormente se fueron mejorando y adaptando algunos aspectos al contexto de la herramienta desarrollada, en base a las sugerencias del cliente.

Otro aspecto importante del desarrollo de la aplicación, fue el estudio y diseño de la forma en que los usuarios tendrían acceso a los datos, porque aunque CONEST tiene un repositorio de datos bien estructurado, éste cuenta con una gran cantidad de tablas y datos normalizados que complican la tarea de elaborar un reporte. Debido a esto, se estudió el modelo de datos actual del sistema CONEST y se analizó la mejor manera de desarrollar un modelo de datos más sencillo, que simplificara el acceso a los datos. En base a todo esto se decidió desarrollar un modelo en base a vistas, donde cada una de ellas recibe el nombre de entidad y agrupa los datos que se encontraban en varias tablas, todo en una sola estructura, simplificando el manejo de los datos y disminuyendo la carga sobre el usuario.

La implantación de la aplicación fue una actividad sencilla en la que se integró el modelo de vistas y el código fuente del generador de reportes al sistema CONEST que se encontraba en producción. Durante esta actividad no hubo complicaciones que acarrearan cambios exhaustivos en la aplicación, logrando de forma satisfactoria el objetivo planteado.

Con la finalidad de corroborar el buen funcionamiento de la aplicación integrada al sistema CONEST se realizaron varias pruebas, las cuales arrojaron resultados positivos y tiempos de respuesta aceptables.

Este Trabajo Especial de Grado permitió al grupo de desarrollo ampliar su experiencia profesional en la creación de aplicaciones Web y en el manejo de grandes volúmenes de datos, ya que en esta actividad se tuvo la oportunidad de tener un cliente, se aplicó una metodología de desarrollo y se trabajó con un lenguaje de programación en el que no se había desarrollado un proyecto de tal envergadura en el pasado, todo con el objetivo de lograr proveerle al cliente una solución acorde a sus necesidades.

RECOMENDACIONES

La aplicación desarrollada en el presente Trabajo Especial de Grado es un generador de reportes que provee a los usuarios la información contenida en el repositorio de datos de una manera concisa. Maneja tres formatos de documentos al momento de exportar los resultados del reporte: PDF, CSV y EXCEL.

Sin embargo existen funcionalidades y características que se podrían añadir o mejorar en la implementación de la aplicación. Estas funcionalidades son:

- Manejo de otros formatos de documentos, como por ejemplo Microsoft Word.

- Soporte para estilos personalizados por el usuario, tales como tipo, tamaño y color de la fuente, permitir al usuario definir su propia plantilla al exportar el reporte a PDF, entre otras.

- Agregar funcionalidades que permitan el envío del reporte al usuario vía correo electrónico.

- Proveer compatibilidad con otros sistemas manejadores de bases de datos distintos a MySQL. Esta aplicación trabaja exclusivamente con MySQL y usa características propias de este, como por ejemplo el mecanismo de acceso y manejo de las vistas, verificación de tipos de datos, entre otros, y esto puede variar de un manejador a otro.

REFERENCIAS

[EMS, 2008] EMS SQL Manager for MySQL. Consultado el 4 de mayo de 2008 de http://www.sqlmanager.net/products/mysql/manager

[Gray, 2007] Gray, J. (2007). Faster CSV Documentation. Recuperado el 17 de noviembre de 2008 de http://fastercsv.rubyforge.org/

[Mañón, 2004] Mañón, E. (2004). Crystal Reports. Recuperado el 3 de mayo de 2008 de http://www.elguille.info/colabora/puntoNET/ElMoreno_CrystalVB.htm

[Márquez, 2008] Márquez, M. & Fernández, W. (2008). *Tópicos para el desarrollo de un Módulo de generación de reportes del sistema CONEST*. Universidad Central de Venezuela.

[MySQL, 2008] MySQL 5.0 Reference Manual. Recuperado el 15 de noviembre de 2008 de http://dev.mysql.com/doc/refman/5.0/es/index.html

[Navicat, 2006] *Navicat MySQL user manual*. (2006). PremiumSoft.

[Spreadsheet, 2009] RubyForge: spreadsheet. Consultado el 14 de enero de 2009 en http://rubyforge.org/projects/spreadsheet/

[Tidwell, 2005] Tidwell, J. (2005). *Designing interfaces*. O' Reilly.

[XML, 2008] REXML. Recuperado el 20 de octubre en http://www.germane-software.com/software/rexml/

[Ziegler, 2005] Ziegler, A. (2005). PDF::Writer for Ruby. Recuperado el 10 de enero de 2009 de http://ruby-pdf.rubyforge.org/pdf-writer/manual/manual.pdf

APÉNDICE

APÉNDICE A

ENCUESTA DE USABILIDAD

Nombre: _____
Fecha: _____

1. ¿El tiempo de respuesta del sistema es adecuado?
   - o Si.
   - o No.
   - o No sabe/ No contesta.

2. ¿Entiendes lo que tienes que hacer con este sistema?
   - o Si.
   - o No.
   - o No sabe/ No contesta.

3. ¿Fue fácil aprender a utilizar el sistema?
   - o Si.
   - o No.
   - o No sabe/ No contesta.

4. ¿La información de ayuda fue útil?
   - o Si.
   - o No.
   - o No sabe/ No contesta.

5. ¿El sistema provee una buena retroalimentación (feedback)?
   - o Si.
   - o No.
   - o No sabe/ No contesta.

6. ¿Trabajar con este sistema es satisfactorio?
   - o Si.
   - o No.
   - o No sabe/ No contesta.

7. ¿La forma como el sistema presenta la información es clara y comprensible?
   - o Si.
   - o No.
   - o No sabe/ No contesta.

8. ¿Este sistema se adapta a tus necesidades?
   - o Si.
   - o No.
   - o No sabe/ No contesta.

9. ¿El funcionamiento del sistema es consistente?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

10. ¿Piensas que puedes realizar tu trabajo de una forma más eficiente utilizando este sistema?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

11. ¿El sistema te permitió revertir alguna acción errónea?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

12. ¿Los mensajes de error son adecuados?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

13. ¿El sistema hace lo que esperas que haga?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

14. ¿El sistema tiene una buena presentación?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

15. ¿Es fácil explorar el sistema?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

16. ¿Consideras que los formatos para la obtención de resultados son idóneos?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

17. ¿Necesitarías el apoyo de un técnico para utilizar nuevamente el sistema?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

18. ¿Las funciones del sistema están bien organizadas?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

19. ¿La organización de la información es clara?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

20. ¿Sientes confianza al utilizar el sistema?
    - o Si.
    - o No.
    - o No sabe/ No contesta.

Sugerencias:

APÉNDICE B

CASO DE ESTUDIO

B.1 Ambiente y especificación de las pruebas

Para realizar las pruebas de la aplicación se seleccionaron seis reportes, de los cuales tres son de tipo Listado y el resto de ellos son de tipo Tabla Cruzada. Para cada uno de estos reportes se indica el enunciado del mismo, se explica la secuencia de pasos que se siguió para generar el reporte, indicando detalles de cada uno de los pasos y mostrando capturas de la interfaz de dichos pasos, y se hace un análisis sobre si los resultados obtenidos fueron los esperados, verificando que los resultados arrojados por la aplicación son los mismos que los obtenidos con la herramienta Crystal Reports XI. Adicionalmente se indica el tiempo empleado en la elaboración del reporte utilizando la aplicación y se realizó la exportación a uno de los formatos soportados.

Para medir los tiempos de elaboración del reporte, el usuario que realiza el reporte tiene conocimientos del modelo de datos de la aplicación y el computador utilizado tiene las siguientes especificaciones:

- Procesador de 8 núcleos de 2.16 GHz.
- Memoria RAM de 8 GB.
- Sistema operativo Linux Debian.
- Sistema manejador de base de datos MySQL Server 5.0.45.
- Navegador Web Mozilla Firefox 3.0.8.

B.2 Prueba 1

B.2.1 Enunciado

Listado de graduandos de la promoción 1 del año 2007 que recibieron premio.

B.2.2 Secuencia de pasos a seguir

a. Para tener acceso al módulo de generación de reportes se debe seleccionar en el menú principal del sistema CONEST la opción “Crear Reportes”.

b. Luego en el primer paso del wizard (figura B-1) el usuario debe ingresar los datos del reporte. Estos datos son:
   - **Nombre:** Es el nombre del reporte que será guardado en el repositorio de datos y el que será asignado al momento de exportar el reporte a algún formato.
   - **Descripción:** Son características y especificaciones del reporte que serán guardadas en el repositorio de datos.
   - **Tipo de reporte por frecuencia:** Se ofrecen dos opciones, “Por solicitud” y “Regular”, el usuario debe seleccionar la que mejor se adapte a su reporte en base a lo explicado en el capítulo 3 del presente Trabajo Especial de Grado.
   - **Forma como desea realizar el reporte:** Se ofrecen dos opciones, “Secuencia de pasos” y “SQL”, el usuario debe seleccionar la opción de su preferencia.

   Para este caso específico el usuario seleccionó “Por solicitud” como frecuencia del reporte y “Secuencia de pasos” como forma para realizar el reporte.

Figura Nº B-1. Registro de datos del reporte de la Prueba 1.

c. Seguidamente, en el segundo paso de la opción “Secuencia de pasos”, el usuario debe indicar el tipo de dato del reporte (figura B-2). Para esta prueba la opción seleccionada fue “Listado”.

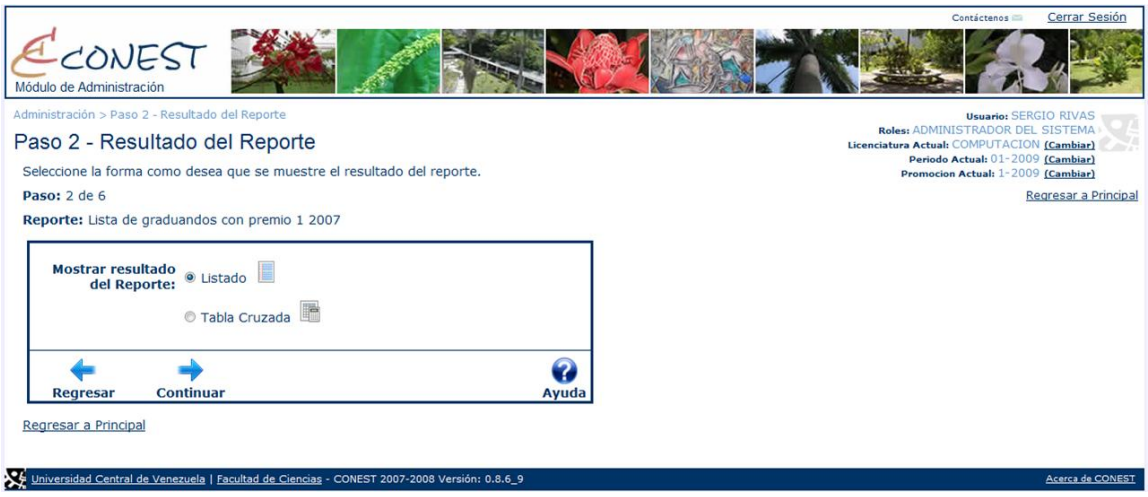


Figura Nº B-2. Registro de tipo del reporte de la Prueba 1.

d. A continuación (figura B-3) se solicita al usuario que seleccione los campos a mostrar en el reporte. Los campos elegidos fueron la cédula del estudiante, el nombre completo, la licenciatura y el nombre del premio recibido.

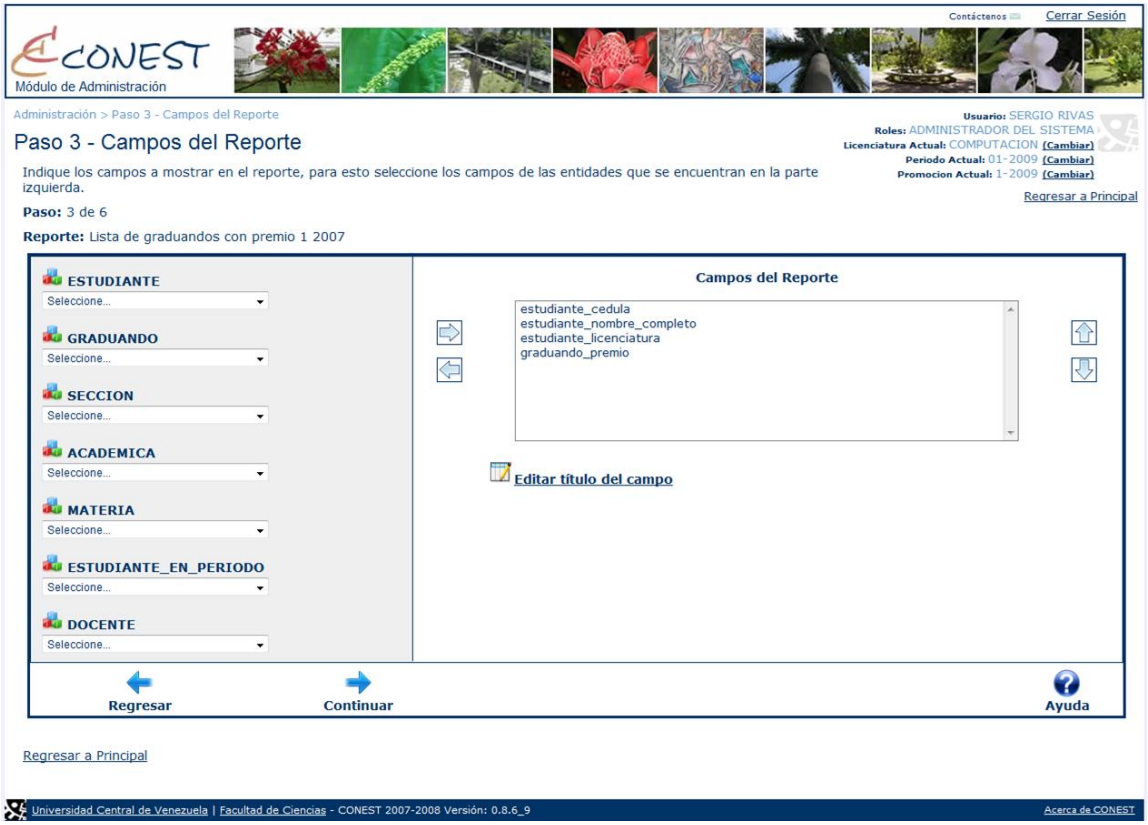


Figura Nº B-3. Selección de campos del reporte de la Prueba 1.

e. Seguido el usuario deberá definir restricciones sobre el reporte que desea generar (figura B-4), en este caso específico la promoción y el año forman parte de las restricciones, así como el nombre del premio debe ser diferente a “sin premio”, para obtener a aquellos estudiantes que obtuvieron algún premio.

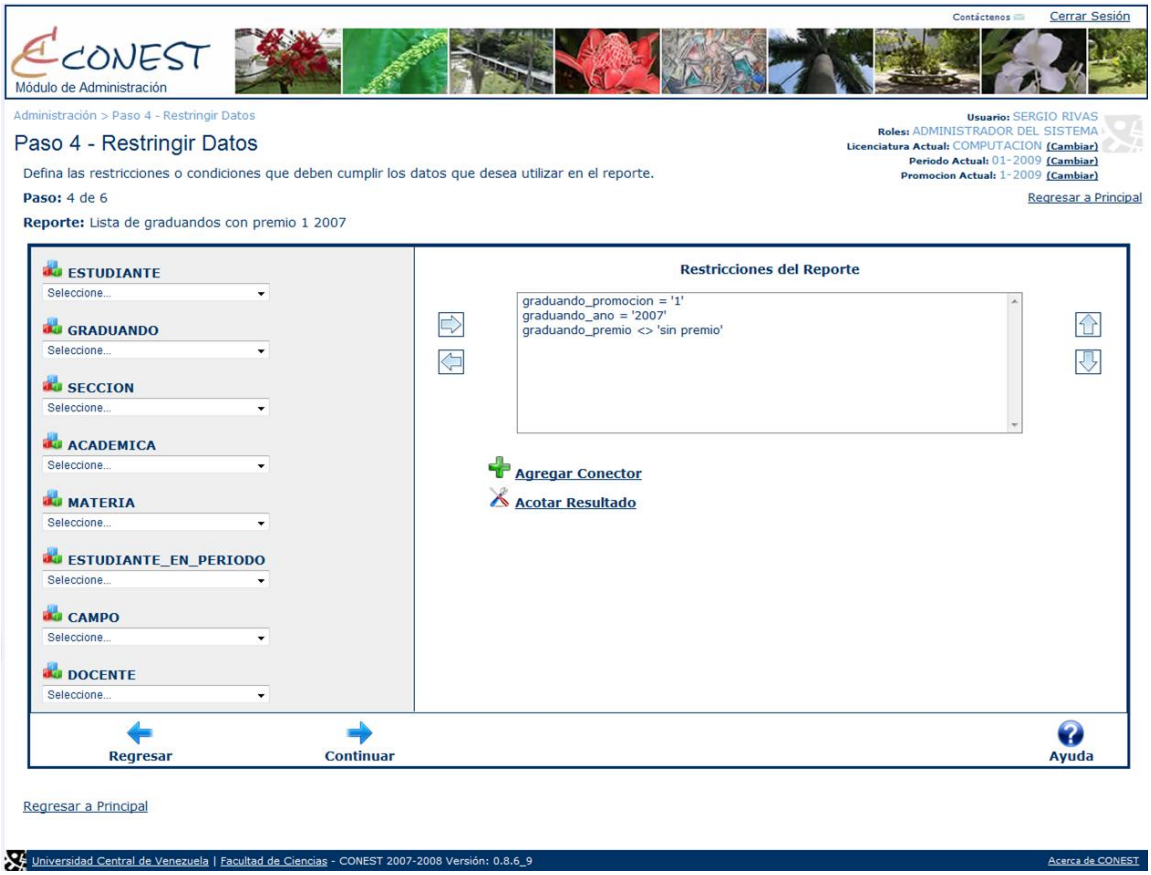


Figura Nº B-4. Definición de restricciones del reporte de la Prueba 1.

f. Luego en el quinto paso, se solicita al usuario de forma opcional que indique el orden del resultado del reporte en base a los campos seleccionados en el paso 3 (figura B-5), en este caso el resultado será ordenado por licenciatura y luego por el nombre del premio para tener una mejor visualización de este.

Figura Nº B-5. Selección de campos de ordenamiento del reporte de la Prueba 1.

g. Finalmente en el sexto paso (figura B-6), se ofrece al usuario la opción de ver la consulta asociada al reporte y de exportarlo a alguno de los formatos soportados por la aplicación. También se brinda la opción de ver el resultado del reporte.

Figura Nº B-6. Resultados del reporte de la Prueba 1.

B.2.3 Análisis de resultados

- **Evaluación del resultado:** Satisfactorio. Se obtuvo el resultado esperado.
- **Tiempo de elaboración del reporte:** Dos minutos cincuenta y siete segundos.
- **Exportación del resultado:** Este reporte fue exportado a formato Excel, este proceso demoró cinco segundos. A continuación (figura B-7) se puede visualizar el resultado en este formato.

	A	B	C	D	E	F
1	UNIVERSIDAD CENTRAL DE VENEZUELA					
2	FACULTAD DE CIENCIAS					
3	COORDINACION ACADEMICA					
4	DIVISION DE CONTROL DE ESTUDIOS					
5						
6	Lista de graduandos con premio 1 2007 17-05-2009 06:59 PM					
7						
8	Nro	estudiante_cedula	estudiante_nombre_completo	estudiante_licenciatura	graduando_premio	
9	1	16889956	PEREIRA RODRIGUEZ ANA CAROLINA	BIOLOGIA	MENCION HONORIFICA	
10	2	16082353	RODRIGUEZ GONCALVES ANDREA	BIOLOGIA	MENCION HONORIFICA	
11	3	15374436	GOMEZ CASTELLANO KEYLA MARIA	BIOLOGIA	MENCION HONORIFICA	
12	4	16877765	CAMPELO MORILLO RIWARD ANTONIO	BIOLOGIA	MENCION HONORIFICA	
13	5	15913035	SUAREZ TORRES JOSE DANIEL	BIOLOGIA	MENCION HONORIFICA	
14	6	15762009	PARRA VALENZUELA ZAFIRO CRISTAL	BIOLOGIA	MENCION HONORIFICA	
15	7	16590760	OTERO MENDOZA REBECA COROMOTO	BIOLOGIA	MENCION HONORIFICA, PREMIO ESPECIAL DE GRADUACION	
16	8	15328971	RUBINO RIVERO RITA ELENA	BIOLOGIA	PREMIO ALTO RENDIMIENTO ACADÉMICO, MENCION HONORIFICA	
17	9	16889196	RIVAS QUIROGA IVANA CRISTINA	COMPUTACION	MENCION HONORIFICA	
18	10	16033272	AMUNDARAIN AROCHA YURIMARA	COMPUTACION	MENCION HONORIFICA	
19	11	16147579	GUZMAN HUERTA RICHARD JOSE	COMPUTACION	PREMIO ALTO RENDIMIENTO ACADÉMICO	
20	12	16857213	PARRA CHACON MAYERLIN CAROLINA	COMPUTACION	PREMIO ESPECIAL DE GRADUACION	
21	13	17100830	PERALTA ARCIA MAYRA ALEJANDRA DE JESUS	FISICA	PREMIO ALTO RENDIMIENTO ACADÉMICO	
22	14	16357261	RODRIGUEZ BRICEÑO PABLO ENRIQUE	FISICA	PREMIO ESPECIAL DE GRADUACION	
23	15	13800074	ROA MARINO EDUARDO	MATEMATICA	PREMIO ALTO RENDIMIENTO ACADÉMICO	
24	16	16300618	PADRON VEZGA ADRIANA COROMOTO	MATEMATICA	PREMIO ESPECIAL DE GRADUACION	
25	17	16683419	MARCANO BRITO FRANCIA RUBETSI	QUIMICA	PREMIO ALTO RENDIMIENTO ACADÉMICO	
26	18	15870342	GARCIA HIDALGO WENDY DEL CARMEN	QUIMICA	PREMIO ESPECIAL DE GRADUACION	
27						

Figura Nº B-7. Resultados del reporte de la Prueba 1 en formato Excel.

B.3 Prueba 2

B.3.1 Enunciado

Listado de estudiantes incursos en reglamento de permanencia para un período académico y año lectivo dado.

Como este es un reporte de tipo Regular, debido a que el período y el año son variables, se tomarán como valores de prueba el período 2 y el año lectivo 2007 para la generación de este reporte.

B.3.2 Secuencia de pasos a seguir

a. En el paso 1 del wizard el usuario indica los datos del reporte tal y como se explicó en la prueba anterior, pero esta vez la frecuencia seleccionada es “Regular” y en el segundo paso el tipo de reporte seleccionado es “Listado”.

b. En el tercer paso se seleccionan los campos a mostrar en el reporte (figura B-8). Para esta prueba los campos seleccionados fueron la cédula, el nombre completo, la licenciatura del estudiante y el estado del reglamento.

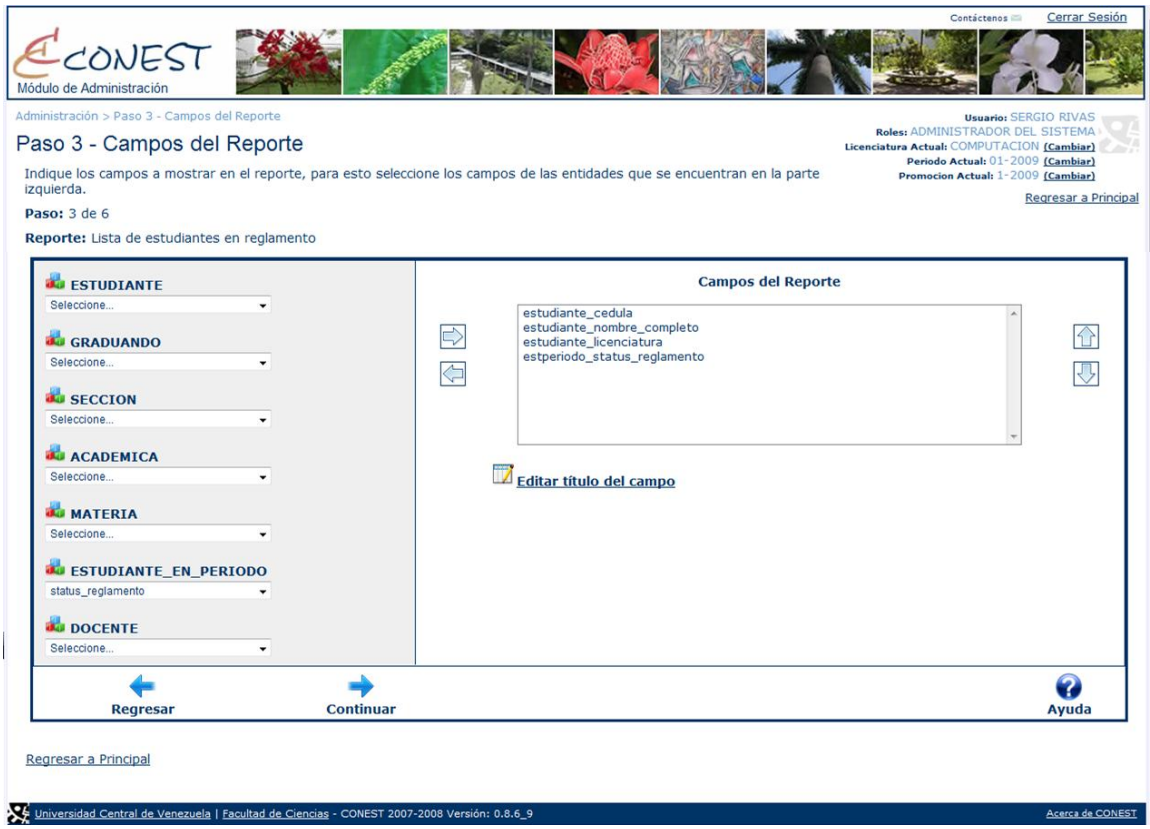


Figura Nº B-8. Selección de campos del reporte de la Prueba 2.

c. Luego se definen las restricciones del reporte (figura B-9), en este caso se indicó que el estado del reglamento del estudiante fuera distinto a “Artículo -–“ y a “Recuperado”, con la finalidad de obtener sólo a los incursos en reglamento, y el período académico y el año lectivo tienen asociado “Valor por definir” debido a que este es un reporte de tipo Regular y estos valores de año y período serán solicitados cada vez que el usuario genere el reporte.

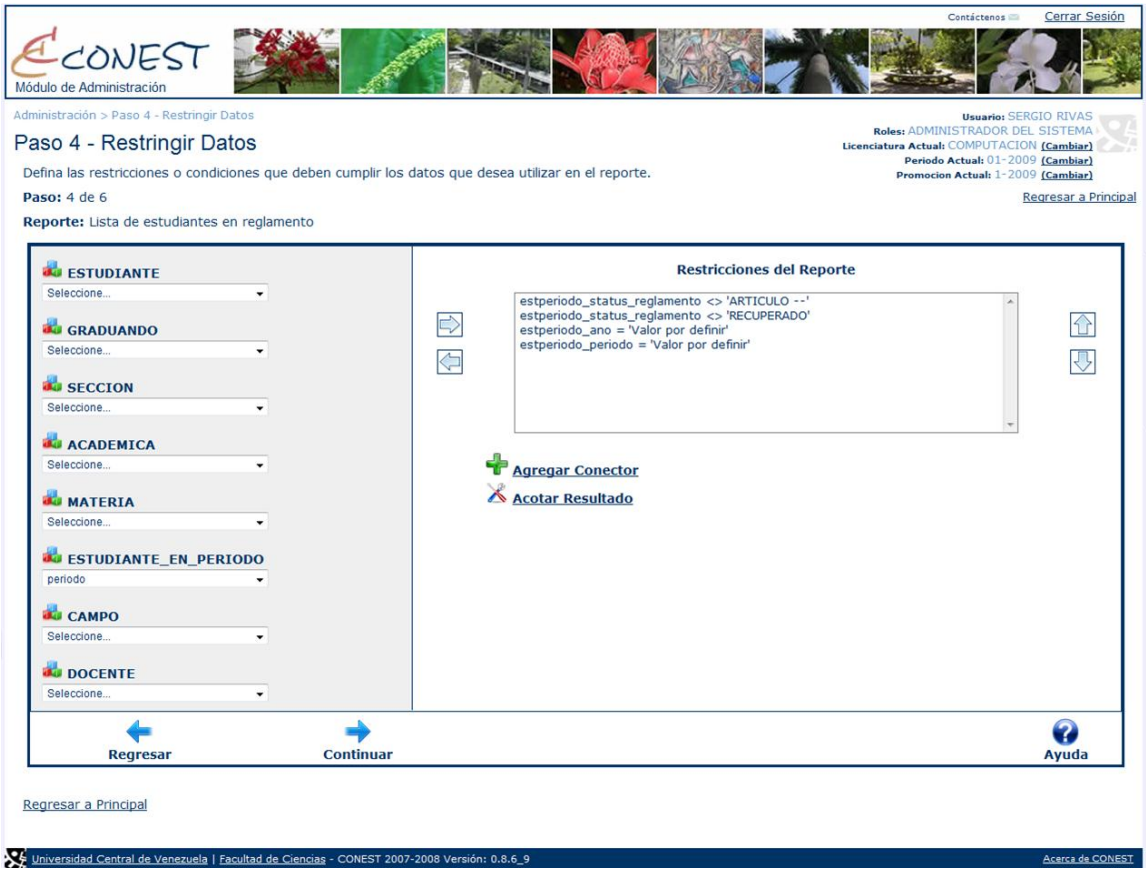


Figura Nº B-9. Definición de restricciones del reporte de la Prueba 2.

d. Seguidamente se indicaron los campos de ordenamiento (figura B-10), para este caso son la licenciatura y el estado del reglamento, y adicionalmente se solicitó calcular la cantidad de estudiantes incursos en cada reglamento.

Figura Nº B-10. Definición de ordenamiento del reporte de la Prueba 2.

e. Por ser un reporte de tipo Regular, a continuación se le solicita al usuario que indique los valores de año lectivo y período académico (figura B-11).

Figura Nº B-11. Registro de valores de las restricciones variables del reporte de la Prueba 2.

f. Una vez hecho esto el usuario puede generar el reporte (figura B-12), se muestra el resultado y a la vez se le ofrece la opción de ver la consulta asociada al reporte, ver los totales producto del conteo solicitado y exportarlo a alguno de los formatos soportados por la aplicación.

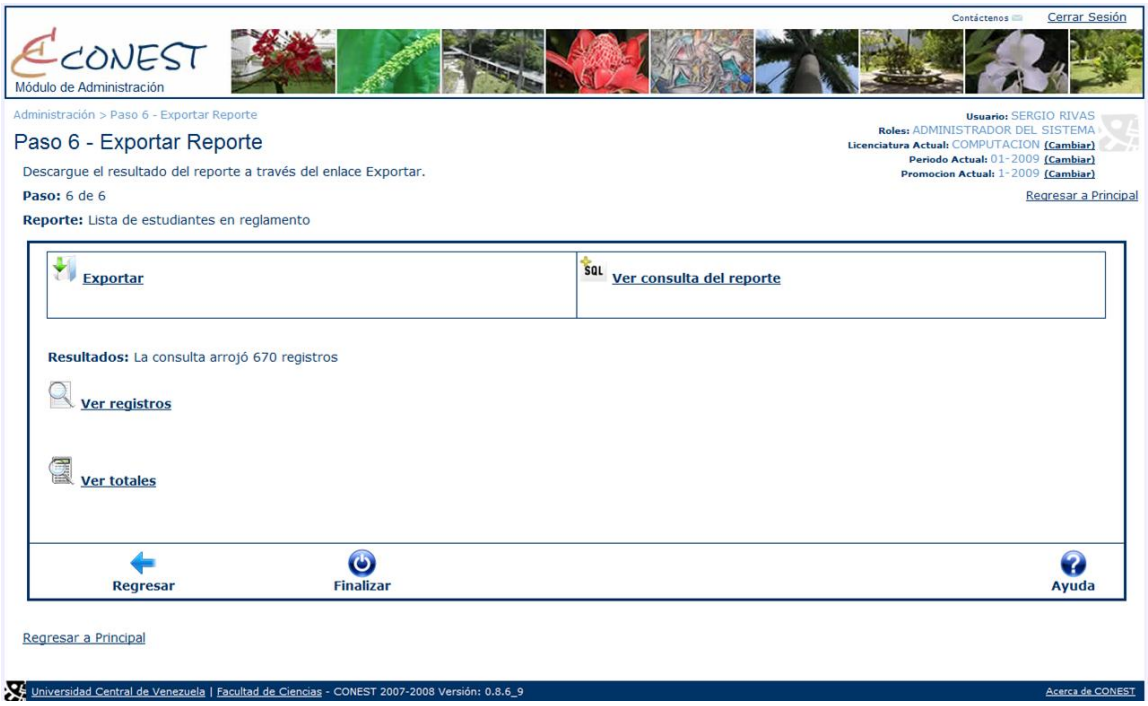


Figura Nº B-12. Resultado del reporte de la Prueba 2.

B.3.3 Análisis de resultados

- **Evaluación del resultado:** Satisfactorio. Se obtuvo el resultado esperado.
- **Tiempo de elaboración del reporte:** Seis minutos con treinta y ocho segundos.
- **Exportación del resultado:** Este reporte fue exportado a formato PDF, se demoró un minuto con cinco segundos debido a la cantidad de registros que contiene el resultado. Se obtuvo un documento con 28 páginas y a continuación (figura B-13) se puede visualizar la primera de ellas con los resultados.

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
COORDINACION ACADEMICA
DIVISION DE CONTROL DE ESTUDIOS

Lista de estudiantes en reglamento

N°	estudiante_cedula	estudiante_nombre_completo	estudiante_licenciatura	estperiodo_status_reglamento
1	19532073	ANGELI MONEO PAOLA CRISTINA	BIOLOGIA	ARTICULO 3
2	18029686	TORRES RODRIGUEZ JHONSON EDUARDO	BIOLOGIA	ARTICULO 3
3	20290497	CARRERO MARTINEZ LESLIE CAROLINA	BIOLOGIA	ARTICULO 3
4	16900937	ALVINS KEDAWEN JESUS ERICK	BIOLOGIA	ARTICULO 3
5	14850659	RINCON PORTAS JUAN DANIEL	BIOLOGIA	ARTICULO 3
6	19153667	BRICEÑO FERNANDEZ RODOLFO MANUEL	BIOLOGIA	ARTICULO 3
7	20484412	NIEVES ACOSTA YEIMY YENIREE	BIOLOGIA	ARTICULO 3
8	18057109	PAVON CUICAS SAMUEL EDUARDO	BIOLOGIA	ARTICULO 3
9	16970836	MEJIA B JOHNDERWIN J	BIOLOGIA	ARTICULO 3
10	18710032	SILVA DUARTES DOUGLAS ALEXANDER	BIOLOGIA	ARTICULO 3
11	15039654	HERNANDEZ DIAZ DELWIS JOSE	BIOLOGIA	ARTICULO 3
12	19191551	MARQUEZ ZAVALA KATHERINE NOHEMI	BIOLOGIA	ARTICULO 3
13	20560037	QUINTERO DIAZ BELEN ANDREINA	BIOLOGIA	ARTICULO 3
14	19561198	CEBALLOS MEDINA RAUL EDUARDO	BIOLOGIA	ARTICULO 3
15	18066844	MOTA JAHEN MILERMIS YETSMAR	BIOLOGIA	ARTICULO 3
16	17023768	VELOSO ARAYA ANDREA DEL PILAR	BIOLOGIA	ARTICULO 3
17	18714035	MOLINA LORETO SOFÍA VICTORIA	BIOLOGIA	ARTICULO 3
18	15343421	IMBRIACO CALZADILLA BARTOLOMEO	BIOLOGIA	ARTICULO 3
19	19202816	QUIROZ A GERARDINA M	BIOLOGIA	ARTICULO 3

Sistema de gestión académica de Control de Estudios CONEST 1/28

Figura Nº B-13. Resultado del reporte de la Prueba 2 en formato PDF.

B.4 Prueba 3

B.4.1 Enunciado

Listado de materias ofertadas por la licenciatura de Computación en el período 1 del año lectivo 2008.

B.4.2 Secuencia de pasos a seguir

a. En el paso 1 el usuario ingresa los datos del reporte, sólo que para este caso selecciona la opción “SQL” como forma de generar el reporte.

b. En el paso 2 de esta opción, se le solicita al usuario que ingrese la consulta asociada al reporte (figura B-14).

Figura Nº B-14. Registro de la consulta del reporte de la Prueba 3.

c. En el siguiente paso (figura B-15), el usuario puede visualizar el resultado del reporte y además se le ofrece la opción de exportar el reporte a cualquiera de los formatos soportados por la aplicación.

Figura Nº B-15. Resultado del reporte de la Prueba 3.

B.4.3 Análisis de resultados

- **Evaluación del resultado:** Satisfactorio. Se obtuvo el resultado esperado.
- **Tiempo de elaboración del reporte:** Dos minutos con treinta y cinco segundos (previamente el usuario había elaborado la consulta del reporte en una herramienta de construcción de consultas).
- **Exportación del resultado:** Este reporte fue exportado a formato CSV, se demoró diez segundos este proceso y a continuación (figura B-16) se puede visualizar la primera parte del resultado utilizando la herramienta Microsoft Excel.

1	materia_codigo	materia_nombre_completo	materia_tipo
2	6211	INTERACCION HUMANO-COMPUTADOR	COMPLEMENTARIA
3	6347	INFORMÁTICA SOCIAL TECNOLOGIAS DE LA INFORMACIÓN Y LA COMUNICACIÓN EN A.L	COMPLEMENTARIA
4	6301	INTRODUCCION A LA INFORMATICA	COMPLEMENTARIA
5	8601	INTRODUCCIÓN A LA COMPUTACIÓN	COMPLEMENTARIA
6	6246	APLICACIONES CON TECNOLOGIA INTERNET II	ELECTIVA
7	6224	INTRODUCCION A LA COMPUTACION GRAFICA	ELECTIVA
8	6332	INNOVACION TECNOLOGICA	ELECTIVA
9	6240	PATRONES DE DISEÑO Y FRAMEWORKS	ELECTIVA
10	6345	ALMACENAMIENTO DE DATOS DE SOPORTE DE DECISIONES	ELECTIVA
11	6231	FUNDAMENTO Y TECNICAS EN COMPUTACION GRAFICA	ELECTIVA
12	6324	SISTEMAS DE BASE DE DATOS ORIENTADO A OBJETO	ELECTIVA
13	6242	ENSEÑANZA ASISTIDA POR COMPUTADOR	ELECTIVA
14	6329	TOPICOS EN INTELIGENCIA ARTIFICIAL	ELECTIVA
15	6031	PROTOCOLO AVANZADO DE REDES	ELECTIVA
16	8601	INTRODUCCIÓN A LA COMPUTACIÓN	ELECTIVA
17	6216	OBJETOS DE APRENDIZAJE: ASPECTOS PEDAGOGICOS Y TECNOLOGICOS	ELECTIVA
18	6226	TECNICAS AVANZADAS DE PROGRAMACION	ELECTIVA
19	6241	DESARROLLO DE APLICACIONES DISTRIBUIDAS	ELECTIVA
20	6211	INTERACCION HUMANO-COMPUTADOR	ELECTIVA
21	6323	CONSTRUCCION DE SISTEMAS DE INFORMACION	ELECTIVA
22	6344	PLANIFICACION ESTRATEGICA DE SISTEMAS	ELECTIVA
23	6133	ALGEBRA LINEAL NUMERICO EN PARALELO	ELECTIVA
24	6024	DISEÑO DE REDES	ELECTIVA
25	6111	TEORIA DE COLAS Y SIMULACION	ELECTIVA
26	6022	SEGURIDAD EN REDES DE COMPUTADORAS	ELECTIVA
27	6028	CALIDAD DE SERVICIOS EN REDES DE COMUNICACIONES	ELECTIVA
28	6311	ADMINISTRACION DE BASES DE DATOS	ELECTIVA
29	6322	DISEÑO DE SISTEMAS DE INFORMACION	ELECTIVA
30	6325	INTELIGENCIA ARTIFICIAL	ELECTIVA
31	6346	SISTEMAS DE BASES DE DATOS DISTRIBUIDAS	ELECTIVA
32	6217	INGENIERIA DEL CONOCIMIENTO	ELECTIVA
33	6212	LENGUAJES Y COMPILADORES	ELECTIVA
34	6045	REDES MOVILES E INALAMBRICAS	ELECTIVA
35	6011	REDES DE COMPUTADORAS	ELECTIVA
36	6321	ANALISIS DE SISTEMAS DE INFORMACION	ELECTIVA
37	6221	APLICACIONES CON LA TECNOLOGIA INTERNET	ELECTIVA

Figura Nº B-16. Resultado del reporte de la Prueba 3 en formato CSV.

B.5 Prueba 4

B.5.1 Enunciado

Obtener por materia la cantidad de aprobados, aplazados, retirados y aprobados por equivalencia en un período académico y año lectivo dado.

Como este es un reporte de frecuencia Regular, debido a que el período académico y el año lectivo son variables, se tomaron como valores de prueba el período 2 y el año 2007 para este ejemplo.

B.5.2 Secuencia de pasos a seguir

a. En el primer paso el usuario ingresa los datos del reporte, selecciona como frecuencia del reporte la opción “Regular” y “Secuencia de pasos” como forma de realizar el reporte.

b. En el segundo paso selecciona la opción “Tabla cruzada”, debido a que esta es la opción que mejor se adapta para el tipo de reporte que se desea elaborar.

c. En el tercer paso el usuario selecciona los campos que deben ir por filas, por columnas y al que se debe aplicar la función de conteo (figura B-17). Para este caso específico se ingresó por fila el nombre de la materia, por columna el estado de la misma (academica_status_materia) y se seleccionó la cédula del estudiante como el campo al que se debe aplicar la función de conteo.

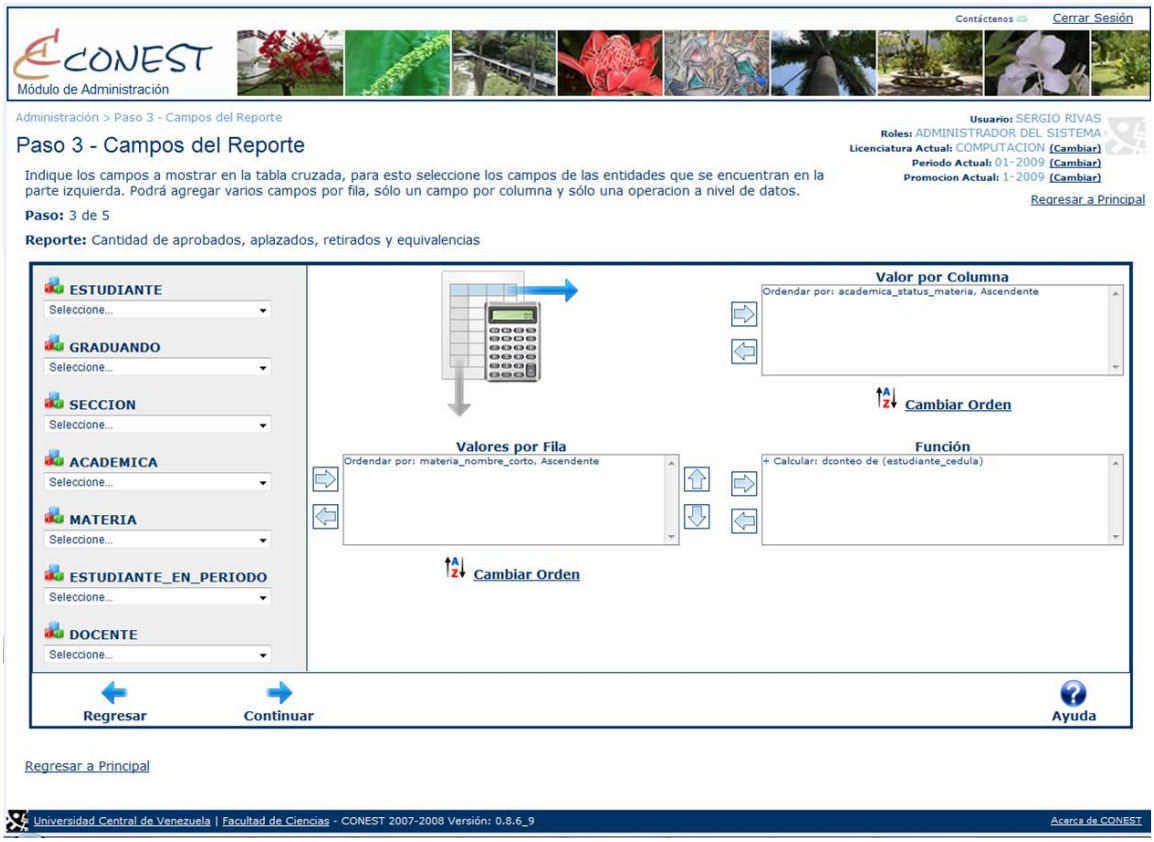


Figura Nº B-17. Selección de campos del reporte de la Prueba 4.

d. Luego el usuario debe definir las restricciones (figura B-18), como este es un reporte de frecuencia regular, el período académico y el año lectivo son de tipo variable, por lo tanto su valor se especifica al momento de generar el reporte.

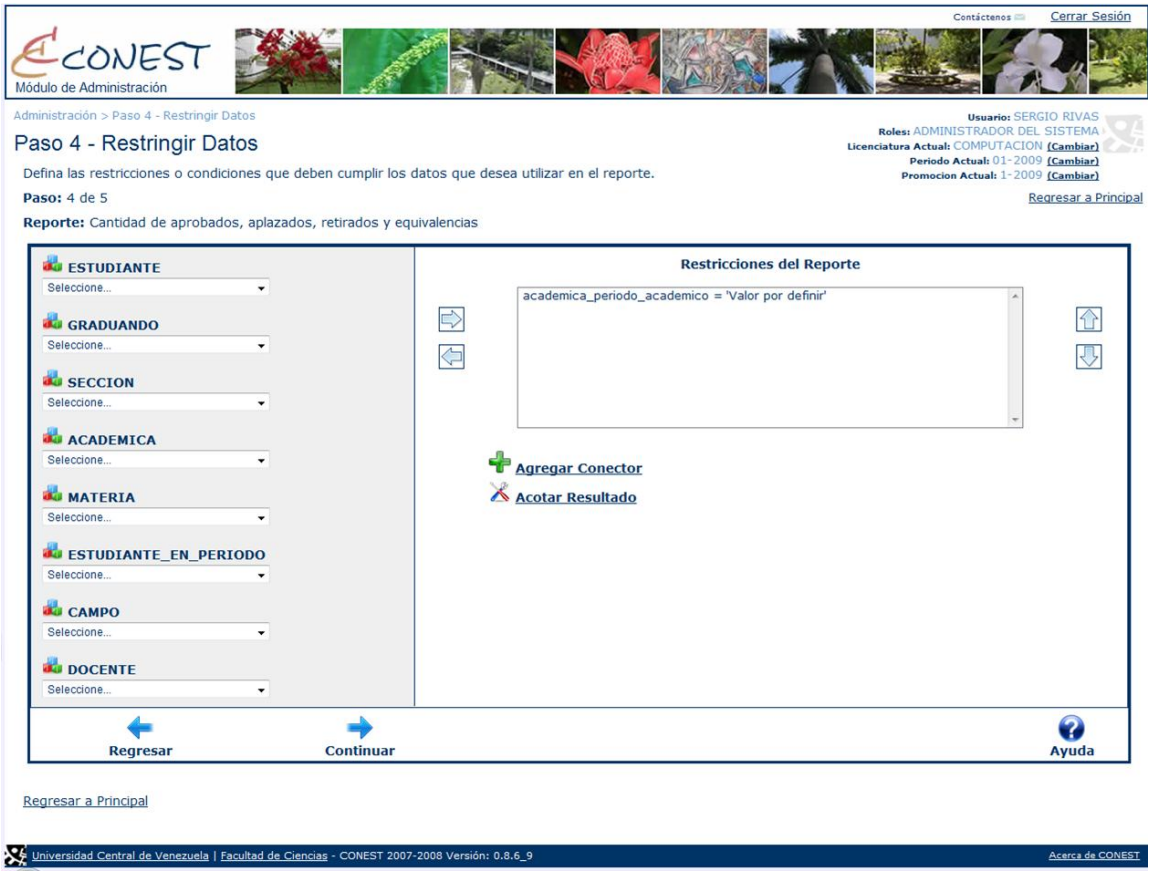


Figura Nº B-18. Definición de restricciones del reporte de la Prueba 4.

e. Seguido se solicita al usuario que ingrese el valor de la restricción variable (figura B-19), en este caso es el valor del campo período académico.

Figura Nº B-19. Registro de valores de las restricciones variables del reporte de la Prueba 4.

f. Una vez hecho esto el usuario puede generar el reporte y además de visualizar el resultado del mismo se le ofrecen las opciones de ver la consulta asociada a este y de exportarlo a alguno de los formatos soportados por la aplicación (figura B-20).

Figura Nº B-20. Resultados del reporte de la Prueba 4.

B.5.3 Análisis de resultados

- **Evaluación del resultado:** Satisfactorio. Se obtuvo el resultado esperado.
- **Tiempo de elaboración del reporte:** Seis minutos con treinta y tres segundos.
- **Exportación del resultado:** Este reporte fue exportado a formato CSV, se demoró un minuto con treinta segundos el proceso debido a las vistas involucradas en la consulta. A continuación (figura B-21) se puede visualizar la primera parte del resultado utilizando la herramienta Microsoft Excel.

1	materia_nombre_corto / academica_status_materia	Sin valor	A	AP	EQ	RET	Total
2	ACT.EN LA ING.DE SOFTWARE	0	9	1	0	1	11
3	ACTIVIDAD CORAL	0	5	0	0	0	5
4	ACTIVIDAD CORAL II	0	4	0	0	1	5
5	ADM. DE BASES DE DATOS	0	51	4	0	4	59
6	AGENTES INTELIGENTES	0	11	1	0	0	12
7	ALG. Y ESTRUT. DE DATOS	0	32	54	0	38	124
8	ALGEBRA I	0	20	13	0	14	47
9	ALGEBRA II	0	10	9	0	1	20
10	ALGEBRA III	0	7	7	0	2	16
11	ALGORITMOS DISTRIBUIDOS	0	11	0	0	1	12
12	ALGORITMOS Y PROGRAMACION	0	87	139	4	41	271
13	ALM.DATOS DE SOPOR. DECIS	0	42	3	0	2	47
14	ANAL.DE REACT QUIMICOS I	0	35	3	0	4	42
15	ANALISIS DE PROCESOS	0	33	4	0	4	41
16	ANALISIS DE SIST. DE INF.	0	16	0	0	9	25
17	ANALISIS FUNCIONAL	0	9	3	0	0	12
18	ANALISIS GEOESTRUCTURAL	0	18	19	0	1	38
19	ANALISIS I	0	10	22	0	9	41
20	ANALISIS II	0	9	11	0	3	23
21	APLIC.CON LA TEC.INTERNET	0	34	4	0	6	44
22	APLIC.CON TEC.INTERNET II	0	26	0	0	0	26
23	APLICAC. CON OBJ.DISTRIB.	0	8	0	0	1	9
24	APRECIACION MUSICAL I	0	38	2	0	0	40
25	APRECIACION MUSICAL II	0	12	2	0	1	15
26	ASTROFISICA III	0	1	0	0	0	1
27	BAS.FIS.TECN.DIAG.TER.MED	0	8	0	0	0	8
28	BASES DE DATOS	0	17	3	1	2	23
29	BIOETICA	0	32	7	0	4	43
30	BIOFISICA Y DINAM.CELULAR	0	2	0	0	3	5
31	BIOGEOGRAFIA	0	4	0	0	2	6
32	BIOLOGIA ANIMAL	0	5	0	0	1	6
33	BIOLOGIA CELULAR	0	5	2	0	1	8
34	BIOLOGIA DE INVERTEBRADOS	0	3	0	0	1	4
35	BIOLOGIA VEGETAL	0	56	13	0	6	75
36	BIOQUIMICA GENERAL	0	53	28	0	7	88
37	CALCULO CIENTIFICO	0	60	38	0	33	131

Figura Nº B-21. Resultados del reporte de la Prueba 4 en formato CSV.

B.6 Prueba 5

B.6.1 Enunciado

Obtener la cantidad de estudiantes que han ingresado a partir del año 2000, indicando el período en el que ingresaron y el tipo de ingreso.

B.6.2 Secuencia de pasos a seguir

a. En el primer paso el usuario ingresa los datos del reporte y en el segundo paso selecciona la opción “Tabla Cruzada”, tal y como se explicó en la prueba anterior.

b. Luego el usuario selecciona el año y el período de ingreso como los campos a mostrar por filas, el tipo de ingreso como el campo a mostrar por columna y la cédula del estudiante como el campo al que se va a aplicar la función de conteo (figura B-22).

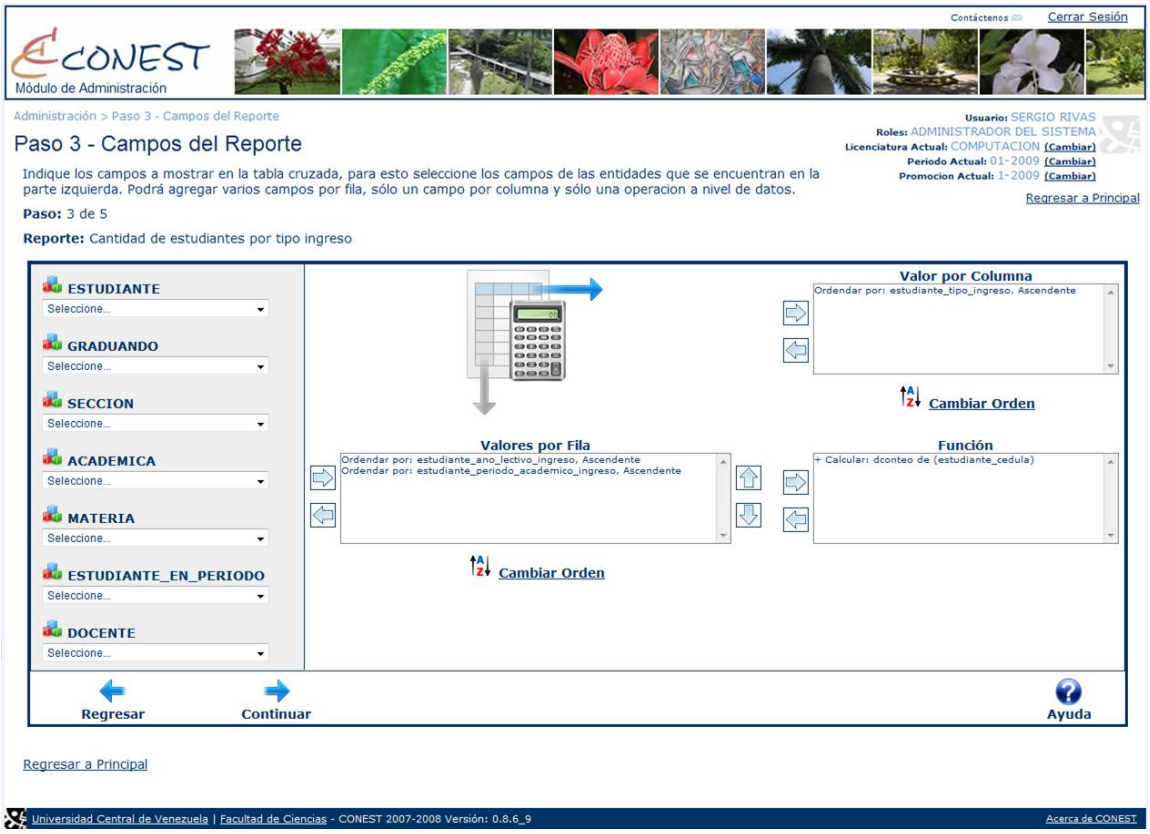


Figura Nº B-22. Selección de campos del reporte de la Prueba 5.

c. Seguidamente el usuario define como restricción que el año de ingreso debe ser mayor o igual a 2000 para calcular la cantidad de estudiantes a partir de esa fecha (figura B-23).

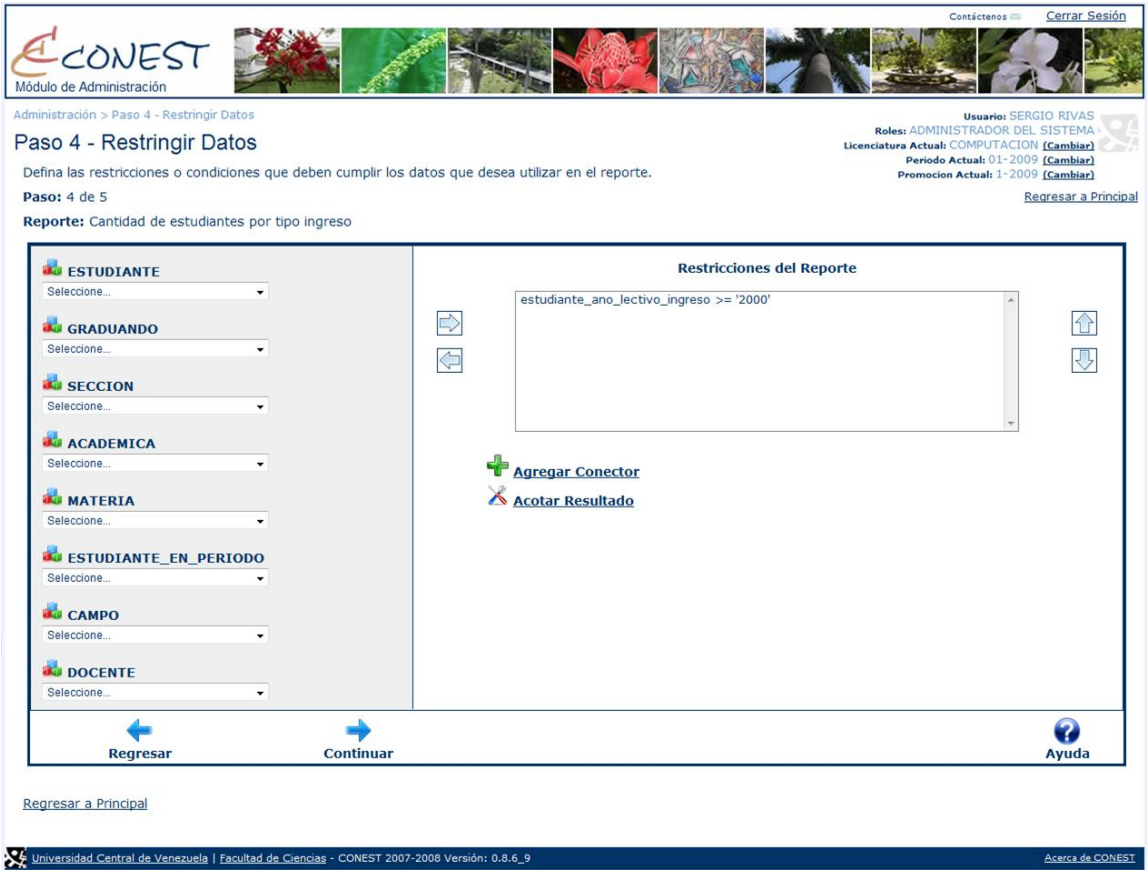


Figura Nº B-23. Definición de restricción del reporte de la Prueba 5.

d. Finalmente se ofrece al usuario la opción de ver la consulta asociada al reporte y de exportarlo a alguno de los formatos soportados por la aplicación. También se brinda la opción de ver el resultado del reporte (figura B-24).

Figura Nº B-24. Resultados del reporte de la Prueba 5.

B.6.3 Análisis de resultados

- **Evaluación del resultado:** Satisfactorio. Se obtuvo el resultado esperado.
- **Tiempo de elaboración del reporte:** Cuatro minutos con catorce segundos.
- **Exportación del resultado:** Este reporte fue exportado a formato Excel, este proceso demoró cinco segundos y a continuación (figura B-25) se puede visualizar el resultado.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	UNIVERSIDAD CENTRAL DE VENEZUELA																	
2	FACULTAD DE CIENCIAS																	
3	COORDINACION ACADEMICA																	
4	DIVISION DE CONTROL DE ESTUDIOS																	
5																		
6		Cantidad de estudiantes por tipo de ingreso 28-04-2009 12:57																
7																		
8	estudiant		Sin valor	ART. 18: C	ART. 23: C	ART. 24: C	ART. 25: D	ART. 26: B	CNU - OP	CURSO IN	EQUIVALE	ESTUDIO	ESTUDIO	PRUEBA I	RES. 158:	SAMUEL	VIA EXCE	Total
9	2000	01	0	20	2	1	5	0	112	104	0	1	0	81	17	0	1	344
10		02	0	52	0	0	6	0	178	2	2	0	0	287	29	1	2	557
11		Total 2000																897
12	2001	01	0	11	0	2	10	0	66	120	9	1	0	71	23	0	12	324
13		02	0	60	2	0	5	0	165	0	0	0	0	234	2	2	36	506
14		Total 2001																830
15	2002	01	0	11	0	0	13	2	59	0	2	0	0	103	42	0	32	264
16		02	0	51	2	0	8	2	185	2	2	0	0	346	9	18	5	630
17		Total 2002																894
18	2003	01	0	4	0	0	10	0	46	0	0	0	0	76	1	1	1	139
19		02	0	42	1	0	12	0	206	0	1	1	0	318	24	6	0	613
20		Total 2003																752
21	2004	01	0	2	1	0	2	1	73	0	3	0	0	77	6	0	3	168
22		02	0	33	0	1	12	7	187	1	1	0	0	307	8	6	0	563
23		I	0	0	0	0	0	0	1	0	2	1	0	0	1	0	0	5
24		Total 2004																736
25	2005	01	0	1	0	0	1	0	23	0	0	1	0	26	5	0	0	57
26		02	0	27	0	0	10	4	223	0	1	0	0	336	6	8	13	628
27		Total 2005																681
28	2006	01	0	2	0	0	1	1	29	0	0	1	0	6	0	0	18	58
29		02	0	33	1	1	11	0	297	0	1	0	0	362	5	10	41	762
30		I	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
31		Total 2006																821
32	2007	01	0	2	2	0	2	0	25	0	6	0	0	6	20	0	10	73
33		02	0	28	0	0	3	3	348	0	8	0	0	353	6	5	21	770
34		Total 2007																842
35	2008	01	0	17	0	1	0	1	43	0	2	0	0	6	5	0	1	76
36		02	0	95	1	0	7	0	255	1	4	0	0	372	5	5	1	746
37		Total 2008																821

Figura Nº B-25. Resultado del reporte de la Prueba 5 en formato Excel.

B.7 Prueba 6

B.7.1 Enunciado

Obtener por cada licenciatura la cantidad de estudiantes según su estado en el sistema, si están en artículo 3, artículo 6, artículo 7, etc., en un año lectivo y período académico dado.

Como este es un reporte de tipo Regular, debido a que el período académico y el año lectivo son variables, se tomaron como valores el período 2 y el año lectivo 2007 para este ejemplo.

B.7.2 Secuencia de pasos a seguir

a. En el primer paso el usuario ingresa los datos del reporte y selecciona como frecuencia del reporte la opción “Regular” y en el segundo paso selecciona la opción “Tabla cruzada”.

b. Luego el usuario selecciona el atributo licenciatura para mostrarlo por filas, el atributo estado del reglamento para mostrarlo por columna y la cédula del estudiante como el campo al que se debe aplicar la función de conteo (figura B-26).

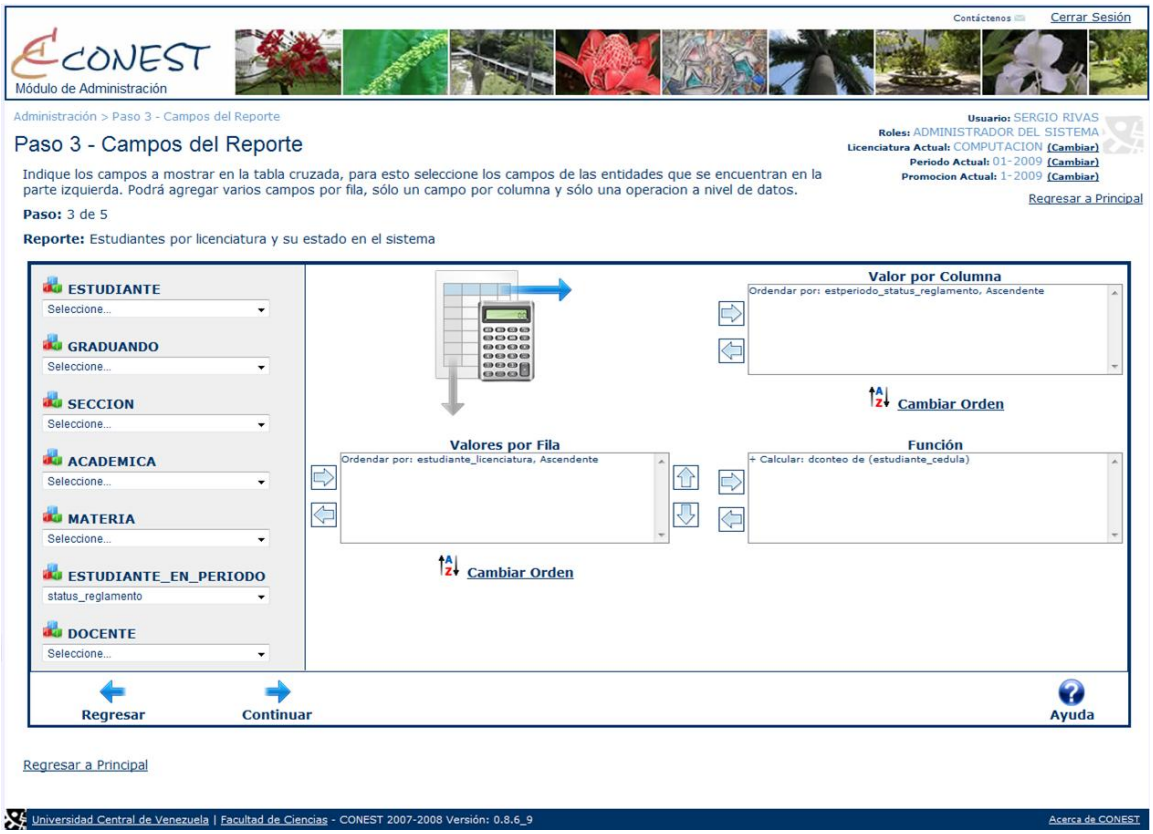


Figura Nº B-26. Selección de campos del reporte de la Prueba 6.

c. El año lectivo y el período académico son definidos como restricciones variables, a los que se les asigna valor en el siguiente paso (figura B-27).

Figura Nº B-27. Definición de valores de las restricciones del reporte de la Prueba 6.

d. Finalmente se ofrece al usuario la opción de ver la consulta asociada al reporte y de exportarlo a alguno de los formatos soportados por la aplicación. También se brinda la opción de ver el resultado del reporte (figura B-28).

Figura Nº B-28. Resultados del reporte de la Prueba 6.

B.7.3 Análisis de resultados

- **Evaluación del resultado:** Satisfactorio. Se obtuvo el resultado esperado.
- **Tiempo de elaboración del reporte:** Cuatro minutos treinta y cinco segundos.
- **Exportación del resultado:** Este reporte fue exportado a formato PDF, este proceso demoró quince segundos y a continuación (figura B-29) se puede visualizar el resultado.

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
COORDINACION ACADEMICA
DIVISION DE CONTROL DE ESTUDIOS

Estudiantes por licenciatura y su estado en el sistema

	ARTICULO --	ARTICULO 3	ARTICULO 6	ARTICULO 7	RECUPERADO	Total
BIOLOGIA	544	93	12	2	6	657
COMPUTACION	876	170	27	8	37	1118
FISICA	250	77	10	1	27	365
GEOQUIMICA	157	13	0	0	5	175
MATEMATICA	257	115	19	5	23	419
QUIMICA	686	103	11	4	20	824

Figura Nº B-29. Resultado del reporte de la Prueba 6 en formato PDF.