Universidad de las Ciencias Informáticas
Facultad 3

Base de Datos para la Residencia de la UCI

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autores: Leonel de Jesús Castillo Santana

Yaniel Alvarez Sánchez

Tutor: Lic. William Martínez Cortés

Consultante: Dr. C. Pascual Verdecia Vicet

Junio del 2007

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Leonel de Jesús Castillo Santana

_____

Yaniel Alvarez Sánchez

_____

Lic. William Martínez Cortés

_____

Datos de Contacto

Lic. William Martínez Cortés

Licenciado en Ciencias de la Computación, graduado en la Universidad de Oriente en 2005. Durante sus estudios de pregrado perteneció al Grupo Científico Investigativo de Programación de Sistemas de la Universidad de Oriente, donde se dedicó a la investigación y producción en el área de los sistemas embebidos y de tiempo real. Desde el tercer año de su carrera fue declarado alumno de alto aprovechamiento, ajustándosele el plan de estudio al perfil de los sistemas de tiempo real. Se graduó con título de oro y 5.04 de promedio.

Luego de su graduación se incorporó a la Universidad de las Ciencias Informáticas (UCI), en donde ha ejercido como profesor de Sistemas Operativos y Seguridad Informática. Simultáneamente al trabajo docente, se desempeñó como Líder del proyecto de Informatización de la Residencia Universitaria.

Dr. C. Pascual Verdecia Vicet

Graduado de Ingeniero de Minas e Ingeniero Civil, Master en Voladura con Explosivos, Doctor en Ciencias Técnicas Y Profesor de Física con 20 años de experiencia, ha investigado y publicado en las ramas de los explosivos y la pedagogía, ha participado en proyectos relacionados con el empleo de explosivos industriales.

Agradecimientos

Agradezco a todas aquellas personas que de una forma u otra han contribuido a mi desarrollo como ser humano de bien. Doy las gracias a mi familia y en especial a mi madre “Cecilia”, a mi padre “Alfonso” y a mi hermano “Luis Manuel” que siempre depositan sus esperanzas en mí y constituyen el aliento imprescindible.

Leonel de Jesús Castillo Santana

Primero que todo quisiera agradecerle a la revolución por darme la oportunidad de estudiar en una universidad como la UCI, institución forjada a calor de la batalla de ideas. Agradecer el apoyo moral y espiritual que me han dado mis padres que representan para mi el sentido total de mi vida, también agradecer la ayuda incondicional de mi tutor y consultante por ser tan concernientes en cada decisión con respecto a mi formación como profesional en Ciencias Informáticas.

Yaniel Alvarez Sánchez

Resumen

Actualmente la Informática constituye un factor muy importante para el desarrollo económico y social de todos los países. Como respuesta a la creciente complejidad de las aplicaciones informáticas y a las necesidades de almacenamiento, seguridad y gestión de la información que persiste han surgido en relativamente poco tiempo los Sistemas de Bases de Datos.

La Universidad de las Ciencias Informáticas es la institución académica más joven de Cuba con el objetivo de graduar profesionales de la Informática altamente calificados que repercutan positivamente en la economía del país. En esta institución se aplican las técnicas más avanzadas en la construcción de software.

En la Residencia de la Universidad de las Ciencias Informáticas se afrontan problemas debido a que la gestión de la información se realiza manualmente en su gran parte y es archivada en formato duro.

Debido a esa problemática la Facultad 3 de la UCI creó un proyecto productivo con el objetivo de implementar un sitio web dinámico en Plone, soportado sobre la plataforma de desarrollo Zope y Python como lenguaje de programación de alto nivel para garantizar la gestión automatizada de la información en su Residencia.

Es interés del cliente que la base de datos sea independiente de la aplicación por lo que no es factible utilizar la base de datos integrada de Zope.

Éste trabajo de diploma es el resultado del diseño de una base de datos en PostgreSQL para los módulos priorizados del proyecto. También se documenta la implementación de la base de datos para el módulo priorizado Reportes de Indisciplinas, así como el diseño del Acceso a Datos para dicho módulo y su integración con la plataforma Zope.

La base de datos está en PostgreSQL porque ofrece muchas ventajas. Entre ellas podemos citar:

- Poder instalar un número ilimitado de veces sin temor de sobrepasar la cantidad de licencias, la principal preocupación de muchos proveedores de bases de datos comerciales.
- Velocidad y rendimiento excepcionales.
- Confiabilidad a toda prueba.
- Seguridad de primera clase.
- Diseño altamente escalable.
- Bajo Costo Total de Operación (TCO).
- Cumplimiento de los estándares ANSI.

Se realizaron una serie de tareas tales como el estudio de PostgreSQL y las herramientas asociadas, la instalación del software necesario, la obtención de los requisitos funcionales y no funcionales que repercutieron en la confección del modelo de datos, así como investigaciones y pruebas de compatibilidad entre las tecnologías Plone/Zope, Phyton y PostgreSQL.

En este trabajo de diploma se exponen las tendencias, técnicas y tecnologías que son utilizadas en la actualidad en los Sistemas de Bases de Datos. Después se realiza la descripción y el análisis de la solución propuesta. Finalmente se exponen una valoración de los resultados obtenidos, las conclusiones y las recomendaciones.

Indice

Páginas

Introducción

Desde hace varias décadas el impacto de la Informática en la vida del hombre es cada vez más evidente. Para los países del mundo es una necesidad impostergable adoptar las Tecnologías de la Informática y las Telecomunicaciones (TICs) para alcanzar sus perspectivas de desarrollo. La producción y consumo de aplicaciones informáticas está marcando pauta en las relaciones económica a nivel internacional.

En Cuba, desde el triunfo de su Revolución y a pesar de ser un país subdesarrollado, se realizan grandes esfuerzos por propiciar e incentivar el desarrollo de la informática. El ejemplo más contundente lo constituye la Universidad de las Ciencias Informáticas (UCI), institución forjada al calor de la Batalla de Ideas, y que posee un elevado compromiso social.

Actualmente la UCI está en la necesidad de automatizar las actividades que ocurren en su interior para lograr la productividad que se requiere y estar a la altura de lo que necesita la sociedad cubana. En este proceso están involucradas todas las áreas de la institución incluyendo a la Residencia. La utilización de software libre se considera muy importante.

En la Residencia de la UCI la gestión de la información se realiza manual en su gran parte y es archivada en formato duro. Esto conlleva a que el nivel de errores sea considerable y muchas veces no detectable. Los sistemas informáticos utilizados brindan una cantidad insuficiente de funcionalidades y, en algunos casos, se dan problemas de incompatibilidad entre ellos. También sucede que los programas utilizados son propietarios y no están debidamente registrados. Esta situación impide que se garanticen todas las necesidades de almacenamiento, disponibilidad, integridad, control, autenticidad, consistencia y auditoria de la información.

Para dar respuesta a esa problemática la Facultad 3 de la UCI tuvo la iniciativa y el compromiso de crear un proyecto productivo con el objetivo de implementar un sitio web para el área de la Residencia que garantice la gestión automatizada de los procesos que ocurren en su interior.

El proyecto está organizado en cinco módulos: *Aseo y Lavandería, Personal, Trabajo Educativo, Alojamiento, Medios Básicos*. En el módulo *Trabajo Educativo* existe un subsistema nombrado *Reportes de Indisciplinas*. La aplicación se construirá básicamente con tecnología de software libre lo cual está en correspondencia con la política trazada por la universidad. El proyecto ha priorizado el diseño de los módulos Reportes de Indisciplinas, Alojamiento y Medios Básicos.

Este trabajo expone la solución dada por los diseñadores de base de datos del proyecto antes mencionado al problema de que no existe el diseño de BD para los módulos priorizados, así como la estrategia de acceso a la misma desde Zope.

El objetivo general es diseñar la BD para los módulos priorizados del proyecto y la estrategia de acceso a dicha BD desde Zope.

Los objetivos específicos consisten en:

1. Diseñar la BD para los módulos Reportes de Indisciplinas, Alojamiento y Medios Básicos.

2. Implementar la BD para el módulo Reportes de Indisciplinas.

3. Diseñar el Acceso a Datos para el módulo Reportes de Indisciplinas.

El objeto de estudio es la gestión automatizada de la información mediante BD relacionales.

El campo de acción es la gestión automatizada de la información en la Residencia de la UCI mediante BD en PostgreSQL.

La hipótesis que se plantea es que si se diseña una BD para PostgreSQL así como su acceso desde Zope entonces se podrá viabilizar la gestión automática de la información en la aplicación web de la Residencia de la UCI.

Para darle cumplimiento a los objetivos será necesario realizar una serie de actividades:

- Estudio y análisis de PostgreSQL y las herramientas asociadas.
- Estudio sobre el acceso desde la plataforma Zope a BD externas.
- Instalación del software necesario.
- Análisis de los requisitos funcionales y no funcionales que incidirán en el diseño de la BD.
- Confección de los modelos lógico y físico de la BD.
- Validación teórica del diseño:

  Normalización de la BD.
  Análisis de redundancia de información.
- Implementación de la BD con las tablas, funciones, disparadores, vistas, usuarios y demás objetos que se necesiten.
- Diseño del acceso a datos para el módulo Reportes de Indisciplinas.
- Análisis de optimización de las consultas.
- Pruebas a la BD con las herramientas y consultas seleccionadas.
- Valoración de los resultados y propuesta de iteración en el diseño.

- Este trabajo de diploma cuenta con 3 capítulos. El capítulo 1 se refiere a la fundamentación teórica de la investigación. En él se exponen las tendencias, técnicas y tecnologías que son utilizadas en la actualidad en los Sistemas de Bases de Datos y que son la vía para la solución del problema. El capítulo 2 esta dedicado a la descripción y análisis de los aspectos del diseño de la BD y del acceso a datos para el módulo Reportes de Indisciplinas. En dicho capítulo se incluyen el diagrama Entidad-Relación, la descripción de las tablas; así como la estrategia a seguir para el acceso a la BD desde la plataforma Zope mediante scripts Python. El último capítulo se refiere a las actividades de validación teórica y funcional del diseño. También se hace una valoración de los resultados obtenidos, se dan las conclusiones y recomendaciones del trabajo, así como la bibliografía consultada y el glosario de términos.

Capítulo 1. Fundamentación Teórica.

Las BD surgen en el año 1960 aunque no se empiezan a usar hasta el año 1971. Estas fueron creadas con el objetivo de solucionar los problemas que existían con el uso en las aplicaciones software de los ficheros, como medio de almacenamiento.

En el entorno informático, la gestión de BD ha evolucionado desde ser un servicio para más disponibilidad de la información hasta ocupar un lugar fundamental en los sistemas de información. En la actualidad, un sistema de información será más valioso mientras de mayor calidad sea la BD que lo soporta, la cual resulta a su vez un componente fundamental del mismo, de tal forma que puede llegarse a afirmar que es imposible la existencia de un sistema de información sin una BD que cumpla la función de "memoria", en todas sus acepciones posibles, del sistema.

Las BD almacenan, como su nombre dice, datos. Entre las numerosas definiciones que pueden encontrarse en la bibliografía, pueden escogerse, por su exhaustividad, las siguientes: (*Los sistemas de bases de datos y los SGBD* 2004)

"Colección de datos correspondientes a las diferentes perspectivas de un sistema de información (de una empresa o institución), existentes en algún soporte de tipo físico (normalmente de acceso directo), agrupados en una organización integrada y centralizada en la que figuran no sólo los datos en sí, sino también las relaciones existentes entre ellos, y de forma que se minimiza la redundancia y se maximiza la independencia de los datos de las aplicaciones que los requieren.”

"Una BD es una colección de datos estructurados según un modelo que refleje las relaciones y restricciones existentes en el mundo real. Los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de éstas, y su definición y descripción han de ser únicas estando almacenadas junto a los mismos. Por último, los tratamientos que sufran estos datos tendrán que conservar la integridad y seguridad de éstos. “

La transferencia entre las entidades del mundo real y sus características, y los registros contenidos en una BD correspondientes a esas entidades, se alcanza tras un proceso lógico de abstracción. Al conjunto de tareas involucradas en dicho proceso se le llama *diseño de la BD*.

Existen muchas formas de organizar las BD, pero hay un conjunto de objetivos generales que deben cumplir todos los Sistemas de Bases de Datos (SBD), de modo que faciliten el proceso de diseño de aplicaciones y que los tratamientos sean más eficientes y rápidos, dando la mayor flexibilidad posible a los usuarios.

Objetivos fundamentales de los SBD.

(GARCIA 1999)

1. Independencia de los datos y los programas de aplicación.

Con ficheros tradicionales la lógica de la aplicación contempla la organización de los ficheros y el método de acceso. Por ejemplo, si por razones de eficiencia se utiliza un fichero secuencial indexado, el programa de aplicaciones debe considerar la existencia de los índices y la secuencia del fichero. Entonces es imposible modificar la estructura de almacenamiento o la estrategia de acceso sin afectar el programa de aplicación (naturalmente, lo que se afecta en el programa son las partes de éste que tratan los ficheros, lo que es ajeno al problema real que el programa de aplicación necesita resolver. En un SBD sería indeseable la existencia de aplicaciones y datos dependientes entre sí, por dos razones fundamentales:

- Diferentes aplicaciones necesitarán diferentes aspectos de los mismos datos (decimal o binario).
- Se debe poder modificar la estructura de almacenamiento o el método de acceso según los cambios en el fenómeno o proceso de la realidad sin necesidad de modificar los programas de aplicación (también para buscar mayor eficiencia).

Independencia de los datos: Inmunidad de las aplicaciones a los cambios en la estructura de almacenamiento y en la estrategia de acceso y constituye el objetivo fundamental de los SBD.

2. Minimización de la redundancia.

Con los ficheros tradicionales, se produce redundancia de la información. Uno de los objetivos de los SBD es minimizar la redundancia de los datos. Se dice disminuir la redundancia, no eliminarla, pues, aunque se definen las BD como no redundantes, en realidad existe redundancia en un grado no significativo para disminuir el tiempo de acceso a los datos o para simplificar el método de direccionado. Lo que se trata de lograr es la eliminación de la redundancia superflua.

3. Integración y sincronización de las bases de datos.

La integración consiste en garantizar una respuesta a los requerimientos de diferentes aspectos de los mismos datos por diferentes usuarios, de forma que, aunque el sistema almacene la información con cierta estructura y cierto tipo de representación, debe garantizar entregar al programa de aplicación de datos que solicita y en la forma en que lo solicita.

Está vinculada a la sincronización, que consiste en la necesidad de garantizar el acceso múltiple y simultáneo a la BD, de modo que los datos puedan ser compartidos por diferentes usuarios a la vez. Están relacionadas, ya que lo usual es que diferentes usuarios trabajen con diferentes enfoques y requieran los mismos datos, pero desde diferentes puntos de vista.

Integridad de los datos.

Consiste en garantizar la no contradicción entre los datos almacenados de modo que, en cualquier momento del tiempo, los datos almacenados sean correctos, es decir, que no se detecte inconsistencia entre los datos. Está relacionada con la minimización de redundancia, ya que es más fácil garantizar la integridad si se elimina la redundancia.

4. Seguridad y recuperación.

Seguridad (también llamada protección): Garantizar el acceso autorizado a los datos, de forma de interrumpir cualquier intento de acceso no autorizado, ya sea por error del usuario o por mala intención.

Recuperación: Que el SBD disponga de métodos que garanticen la restauración de las BD al producirse alguna falla técnica, interrupción de la energía eléctrica, etc.

5. Facilidad de manipulación de la información.

Los usuarios de una BD pueden referirse a ella con las solicitudes para resolver muchos problemas diferentes. El SBD debe contar con la capacidad de una búsqueda rápida por diferentes criterios, permitir que los usuarios planteen sus demandas de una forma simple, aislándolo de las complejidades del tratamiento de los ficheros y del direccionado de los datos. Los SBD actuales brindan lenguajes de alto nivel con diferentes grados de facilidad para el usuario no programador que garantizan este objetivo, los llamados sublenguajes de datos.

6. Control centralizado.

Uno de los objetivos más importantes de los SBD es garantizar el control centralizado de la información. Permite controlar de manera sistemática y única los datos que se almacenan en la BD, así como el acceso a ella.

Lo anterior implica que debe existir una persona o conjunto de personas que tenga la responsabilidad de los datos operacionales: el administrador de la BD, que puede considerarse parte integrante del SBD. Entre las tareas del administrador de la BD se encuentran:

- Decidir el contenido informativo de la BD.
- Decidir la estructura de almacenamiento y la estrategia de acceso.
- Garantizar el enlace con los usuarios.
- Definir los chequeos de autorización y procedimientos de validación.
- Definir la estrategia de reorganización de las BD para aumentar la eficiencia del sistema.

Existen otros objetivos que deben cumplir los SBD que en muchos casos dependen de las condiciones o requerimientos específicos de utilización del sistema.

Tipos de Bases de Datos.

(*Base de Datos* 2007)

Las bases de datos se clasifican de acuerdo a sus características y propósitos, por ejemplo:

1. Según la variabilidad de los datos almacenados.

- **Bases de datos estáticas.**

Éstas son BD de sólo lectura, utilizadas primordialmente para almacenar datos históricos que posteriormente se pueden utilizar para estudiar el comportamiento de un conjunto de datos a través del tiempo, realizar proyecciones y tomar decisiones.

- **Bases de datos dinámicas.**

Éstas son BD donde la información almacenada se modifica con el tiempo, permitiendo operaciones como actualización y adición de datos, además de las operaciones fundamentales de consulta. Un ejemplo de esto puede ser la BD utilizada en un sistema de información de una tienda de abarrotes, una farmacia, un videoclub, etc.

2. Según el contenido.

- **Bases de datos bibliográficas.**

Solo contienen un representante de la fuente primaria, que permite localizarla. Un registro típico de una BD bibliográfica contiene información sobre el autor, fecha de publicación, editorial, título, edición, de una determinada publicación, etc. Puede contener un resumen o extracto de la publicación original, pero nunca el texto completo, porque sino estaríamos en presencia de una BD a texto completo (o de fuentes primarias). Como su nombre lo indica, el contenido son cifras o números. Por ejemplo, una colección de resultados de análisis de laboratorio, entre otras.

- **Bases de datos de texto completo.**

Almacenan las fuentes primarias, como por ejemplo, todo el contenido de todas las ediciones de una colección de revistas científicas.

- **Directorios.**

1. Un ejemplo son las guías telefónicas en formato electrónico.
2. Banco de imágenes, audio, video, multimedia, etc.
3. Bases de datos o "bibliotecas" de información Biológica.

Son BD que almacenan diferentes tipos de información proveniente de las ciencias de la vida o médicas. Se pueden considerar en varios subtipos:

- Aquellas que almacenan secuencias de nucleótidos o proteínas.
- Las BD de rutas metabólicas.
- BD de estructura, comprende los registros de datos experimentales sobre estructuras 3D de biomoléculas.
- BD clínicas.
- BD bibliográficas (biológicas).

Modelos utilizados con más frecuencia en las BD.

(*Base de Datos* 2007)

1. Bases de datos jerárquicas.

Éstas son BD que, como su nombre indica, almacenan su información en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol (visto al revés), en donde un nodo padre de información puede tener varios hijos. El nodo que no tiene padres es llamado raíz, y a los nodos que no tienen hijos se los conoce como hojas. Las BD jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento. Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.

2. Base de datos de red.

Éste es un modelo ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de nodo: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico). Fue una gran mejora con respecto al modelo jerárquico, ya que ofrecía una solución eficiente al problema de redundancia de datos; pero, aun así, la dificultad que significa administrar la información en una BD de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales.

3. Base de datos relacional.

Éste es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postulados sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de BD. Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Pese a que ésta es la teoría de las BD relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar. Esto es pensando en cada relación como si fuese una tabla que está compuesta por registros (las filas de una tabla), que representarían las tuplas, y campos (las columnas de una tabla).

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la BD. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información.

El lenguaje más habitual para construir las consultas a BD relacionales es SQL, Structured Query Language o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de BD relacionales.

Durante su diseño, una BD relacional pasa por un proceso al que se le conoce como normalización de una BD. Durante los años '80 (1980-1989) la aparición de dBASE produjo una revolución en los lenguajes de programación y sistemas de administración de datos. Aunque nunca debe olvidarse que dBase no utilizaba SQL como lenguaje base para su gestión.

4. Base de datos orientada a objetos.

Este modelo, bastante reciente, y propio de los modelos informáticos orientados a objetos, trata de almacenar en la BD los objetos completos (estado y comportamiento).

Una BD orientada a objetos es una BD que incorpora todos los conceptos importantes del paradigma de objetos:

- Encapsulación: Propiedad que permite ocultar la información al resto de los objetos, impidiendo así accesos incorrectos o conflictos.
- Herencia: Propiedad a través de la cual los objetos heredan comportamiento dentro de una jerarquía de clases.
- Polimorfismo: Propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos.

En las BD orientadas a objetos, los usuarios pueden definir operaciones sobre los datos como parte de la definición de la BD. Una operación (llamada función) se especifica en dos partes. La interfaz (o signatura) de una operación incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros). La implementación (o método) de la operación se especifica separadamente y puede modificarse sin afectar la interfaz. Los programas de aplicación de los usuarios pueden operar sobre los datos invocando a dichas operaciones a través de sus nombres y argumentos, sea cual sea la forma en la que se han implementado. Esto podría denominarse independencia entre programas y operaciones.

5. Bases de datos Objeto-Relacionales.

Los modelos de datos relacionales orientados a objetos extienden el modelo de datos relacional proporcionando un sistema de tipos más rico y que se adapte mejor a las nuevas aplicaciones, que requieren guardar un tipo de datos más complejos. Permiten que los atributos de las tuplas tengan tipos complejos. Además, la BD es relacional por lo que conserva su rapidez y eficiencia, y permite hacer uso de nuevos elementos, como las relaciones de herencia, que modelan los objetos que serán guardados en dicha BD, por lo que el analista y el diseñador ven un modelo orientado a objetos.

6. Bases de datos documentales.

Permiten la indexación a texto completo, y en líneas generales realizar búsquedas más potentes. Tesaurus es un sistema de índices optimizado para este tipo de BD.

7. Base de datos deductivas.

Un SBD deductivo, es un SBD pero con la diferencia de que permite hacer deducciones a través de inferencias. Se basa principalmente en reglas y hechos que son almacenados en la BD. También las BD deductivas son llamadas BD lógica, a raíz de que se basan en lógica matemática.

8. Gestión de bases de datos distribuida

La BD está almacenada en varias computadoras conectadas en red. Surgen debido a la existencia física de organismos descentralizados. Esto les da la capacidad de unir las BD de cada localidad y acceder así a distintas universidades, sucursales de tiendas, etc.

El lenguaje SQL.

(*SQL* 2007)

En la actualidad el SQL es el estándar *de facto* de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio.

Los orígenes del SQL están ligados a los de las BD relacionales. El SQL es un lenguaje de acceso a BD que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos. Es un lenguaje declarativo de *alto nivel* o de *no procedimiento*, que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación. El SQL proporciona una rica funcionalidad más allá de la simple consulta (o recuperación) de datos. Asume el papel de lenguaje de definición de datos (LDD), lenguaje de definición de vistas (LDV) y lenguaje de manipulación de datos (LMD). Además permite la concesión y denegación de permisos, la implementación de restricciones de integridad y controles de transacción, y la alteración de esquemas. Las primeras versiones del SQL incluían funciones propias de lenguaje de definición de almacenamiento (LDA) pero fueron suprimidas en los estándares más recientes con el fin de mantener el lenguaje sólo a nivel conceptual y externo.

En el caso de hacer un uso embebido del lenguaje podemos utilizar dos técnicas alternativas de programación. En una de ellas, en la que el lenguaje se denomina *SQL estático*, las sentencias utilizadas no cambian durante la ejecución del programa. En la otra, donde el lenguaje recibe el nombre de *SQL dinámico*, se produce una modificación total o parcial de las sentencias en el transcurso de la ejecución del programa. La utilización de SQL dinámico permite mayor flexibilidad y mayor complejidad en las sentencias, pero como contra punto obtenemos una eficiencia menor y el uso de técnicas de programación más complejas en el manejo de memoria y variables.

Existe una ampliación de SQL conocida como FSQL (Fuzzy SQL, SQL difuso) que permite el acceso a BD difusas, usando la lógica difusa. Este lenguaje ha sido implementado a nivel experimental y está evolucionando rápidamente.

Los siguientes SGBD usan el SQL como lenguaje estándar para la manipulación de los datos en una BD.

- DB2
- Oracle
- SQL Server
- Sybase ASE
- MySQL
- PostgreSQL
- Firebird

Historia de PostgreSQL.

(*SQL* 2007)

PostgreSQL es un Sistema Gestor de Bases de Datos Objeto Relacional (SGBDOR) y libre, de reconocido prestigio y con una extraordinaria penetración en el mercado debido a que es Software Libre, PostgreSQL como tal tiene algo mas de diez años de desarrollo; tiempo en el que se ha convertido en una referencia para aquellos que buscan un SGBD sólido y confiable sin necesidad de hacer un enorme desembolso de dinero.

PostgreSQL es el último resultado de una larga evolución comenzada con el proyecto Ingres en la Universidad de Berkeley. El líder del proyecto, Michael Stonebraker abandonó Berkeley para comercializar Ingres en 1982, pero finalmente regresó a la academia. Tras su retorno a Berkeley en 1985, Stonebraker comenzó un proyecto post-Ingres para resolver los problemas con el modelo de BD relacional que habían sido aclarados a comienzos de los años 1980. El principal de estos problemas era la incapacidad del modelo relacional de comprender "tipos", es decir, combinaciones de datos simples que conforman una única unidad. Actualmente estos son llamados objetos.

El proyecto resultante, llamado Postgres, era orientado a introducir la menor cantidad posible de funcionalidades para completar el soporte de tipos. Estas funcionalidades incluían la habilidad de definir tipos, pero también la habilidad de describir relaciones las cuales hasta ese momento eran ampliamente utilizadas pero mantenidas completamente por el usuario. En Postgres la BD "comprendía" las relaciones y podía obtener información de tablas relacionadas utilizando reglas.

Comenzando en 1986, el equipo liberó una serie de documentos describiendo la base del sistema y en 1988 poseían un prototipo funcional. La versión 1 fue liberada a un pequeño grupo de usuarios en junio de 1989, seguido por la versión 2 con un sistema de reglas reescrito en junio de 1990. Para la versión 3, liberada en 1991, el sistema de reglas fue reescrito nuevamente, pero también agregó soporte para múltiples administradores de almacenamiento y un sistema de consultas mejorado. Hacia 1993, Postgres había crecido inmensamente en popularidad y poseía una demanda asfixiante de nuevas funcionalidades. Tras liberar la versión 4, la cual era una simple versión de limpieza, el proyecto fue abandonado.

A pesar de que el proyecto Postgres hubiese finalizado oficialmente, la licencia BSD bajo la cual Postgres había sido liberado permitió a desarrolladores de código abierto el obtener una copia del código para continuar su desarrollo.

Algunas de sus principales características.

1. Funciones.

Las funciones permiten subir bloques de código que se ejecuten en el servidor. Estas funciones pueden escribirse en una variedad de lenguajes, algunos de los más importantes son PL/pgSQL, C, C++ y Java.

Puede definirse si las funciones serán ejecutadas con los permisos del llamador o del usuario que definió la función.

2. Alta concurrencia.

Mediante un sistema denominado MVCC (Acceso concurrente multiversión) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

3. Amplia variedad de tipos nativos.

PostgreSQL provee nativamente soporte para:

- Números de precisión arbitraria.
- Texto de largo ilimitado.
- Figuras geómetricas (con una variedad de funciones asociadas)
- Direcciones IP (IPv4 e IPv6).
- Bloques de direcciones estilo CIDR.
- Direcciones MAC.
- Arreglos.

Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS.

4. Otras características.

- Cumple completamente con ACID.
- Cumple con ANSI SQL.
- Llaves Foráneas (foreign keys).
- Disparadores (triggers).
- Reglas.
- Vistas.
- Unicode.
- Secuencias.
- Outer Joins.
- Sub-selects.
- Integridad transaccional.
- Herencia de tablas.
- Tipos de datos y operaciones geométricas.
- Replicación (soluciones comerciales y no comerciales) que permiten la duplicación de bases de datos maestras en múltiples sitios de réplica.
- Una API abierta.
- Interfaces nativas para ODBC, JDBC, C, C++, PHP, Perl, TCL, ECPG, Python y Ruby.

- Soporte nativo SSL.
- Respaldo en caliente.
- Bloqueo a nivel mejor-que-fila.
- Índices parciales y funcionales.
- Autentificación Kerberos nativa.
- Soporte para consultas con UNION, UNION ALL y EXCEPT.
- Extensiones para SHA1, MD5, XML y otras funcionalidades.
- Herramientas para generar SQL portable para compartir con otros sistemas compatibles con SQL.
- Funciones de compatibilidad para ayudar en la transición desde otros sistemas menos compatibles con SQL.

Algunas de sus principales ventajas.

1. Instalación ilimitada.

Es frecuente que las BD comerciales sean instaladas en más servidores de lo que permite la licencia. Algunos proveedores comerciales consideran a esto la principal fuente de incumplimiento de licencia. Con PostgreSQL nadie puede hacer alguna demanda por violar acuerdos de licencia, puesto que no hay costo asociado a la licencia del software.

Esto tiene varias ventajas adicionales:

- Modelos de negocios más rentables con instalaciones a gran escala.
- No existe la posibilidad de ser auditado para verificar cumplimiento de licencia en ningún momento.
- Flexibilidad para hacer investigación y desarrollo sin necesidad de incurrir en costos adicionales de licenciamiento.

2. Soporte técnico.

Además de las numerosas ofertas de soporte por parte de empresas de software libre, exiten importantes comunidades de profesionales y entusiastas de PostgreSQL de las que se pueden obtener beneficios y contribuir.

3. **Ahorros considerables en costos de operación.**

El software ha sido diseñado y creado para tener un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características, estabilidad y rendimiento.

Además de esto, a nivel mundial existen programas de entrenamiento que son reconocidamente mucho más baratos, manejables y prácticos en el mundo real que aquellos de los principales proveedores comerciales.

4. **Estabilidad y confiabilidad.**

En contraste a muchos SBD comerciales es extremadamente común que compañías reporten que PostgreSQL nunca ha presentado caídas en varios años de operación de alta actividad.

5. **Extensible.**

El código fuente está disponible para todos sin costo. Si un equipo de desarrollo necesita extender o personalizar PostgreSQL de alguna manera, pueden hacerlo con un mínimo esfuerzo, sin costos adicionales. Esto es complementado por la comunidad de profesionales y entusiastas de PostgreSQL alrededor del mundo que también extienden PostgreSQL todos los días.

6. **Multiplataforma y variadas herramientas de diseño y administración.**

Existen instaladores de los servidores y clientes PostgreSQL para los sistemas operativos Windows y las distribuciones de Linux. También se dispone de diversas herramientas gráficas de alta calidad para el diseño: Toad, Data Architect, etc.

7. **Diseñado para ambientes de alto volumen.**

Al igual que los principales proveedores de SBD comerciales, PostgreSQL usa una estrategia de almacenamiento de filas llamada MVCC para conseguir una mucha mejor respuesta en ambientes de grandes volúmenes.

El lenguaje de programación Python

El lenguaje de programación Python es un lenguaje de alto nivel muy utilizado en la actualidad por sus atractivas características.

1. Es un lenguaje de propósito general y de código abierto.
2. El código fuente que se obtiene siempre es muy legible y elegante.
3. Es minimalista porque todo aquello innecesario no hay que escribirlo.
4. Es denso porque poco código hace mucho.
5. Soporta objetos y estructuras de datos de alto nivel: strings, listas, diccionarios, etc.
6. Posee múltiples niveles de organizar código: funciones, clases, módulos y paquetes.
7. Existe una infinidad de bibliotecas de utilidades de código abierto.
8. Se puede extender o mejorar su funcionalidad utilizando plugins de C o C++, siguiendo la API para extender o empotrar Python en una aplicación, o a través de otras herramientas.
9. No es necesario declarar constantes y variables antes de utilizarlas.
10. La primera vez que se ejecuta un script de Python se compila y genera bytecode que es luego interpretado.
11. Posee una alta velocidad de desarrollo y buen rendimiento.
12. Se puede utilizar en múltiples plataformas.

Python es ideal:

- Como lenguaje "pegamento" para combinar varios componentes juntos.
- Para llevar a cabo prototipos de sistema.
- Para la elaboración de aplicaciones cliente.
- Para desarrollo web y de sistemas distribuidos.
- Para el desarrollo de tareas científicas, en los que hay que simular y prototipar rápidamente.

Python no es un lenguaje perfecto, no es bueno para:

- Programación de bajo nivel, como programación de drivers y kernels.
- Python no tiene control directo sobre memoria y otras tareas de bajo nivel.
- Aplicaciones que requieren alta capacidad de cómputo.

Capitulo 2: Descripción de la Solución.

Introducción

Este capitulo constituye la parte principal del trabajo. En él se incluyen los requisitos no funcionales de la BD, los diseños lógicos y físicos, así como la descripción de las tablas y otros objetos de la BD para el módulo Indisciplinas. Se describe la estrategia de integración de la BD con la plataforma de desarrollo Zope mediante scripts Python y, finalmente, se documentan las clases que conforman la Capa de Acceso a Datos.

Las herramientas de diseño utilizadas son ER/Studio 7.0 para los modelos de la BD y Rational Rose 2003 para las clases de acceso a los datos. También se utilizan PostgreSQL 8.1 como SGBD y el cliente pgAdminIII 1.4.

Descripción de la Arquitectura de PostgreSQL.

La arquitectura de PostgreSQL es de 3 niveles lo cual está en correspondencia con la propuesta por el grupo ANSI/SPARC. Esta arquitectura se corresponde suficientemente bien con un gran número se sistemas, aunque no se puede asegurar que cualquier SGBD se corresponda exactamente con ella.

La arquitectura de PostgreSQL (Figura 1) se divide en los siguientes niveles generales: *interno, lógico global y externo*.

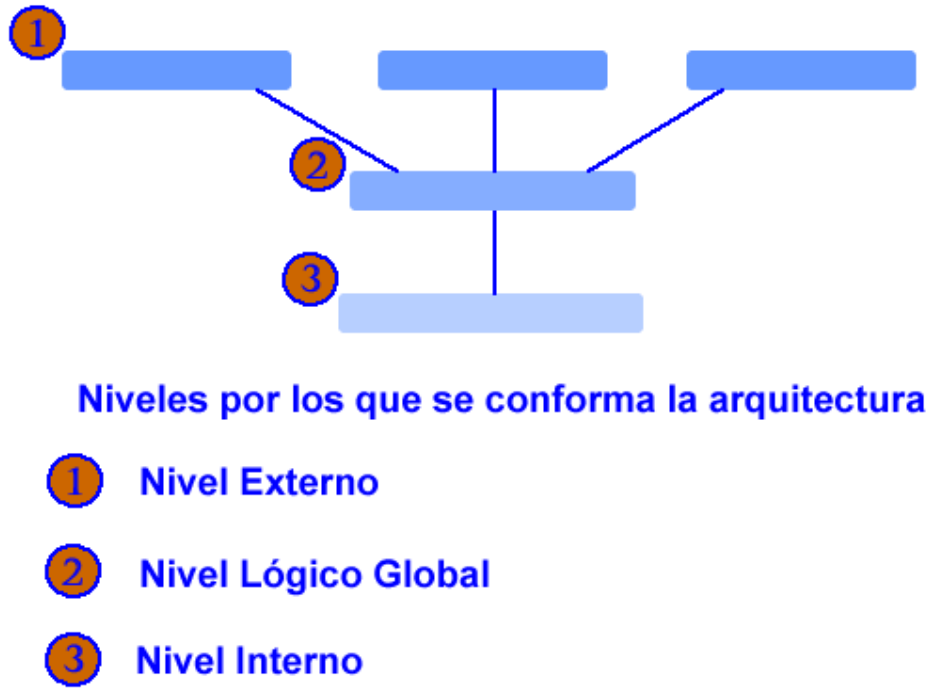

Figura 1. Arquitectura de PostgreSQL.

El nivel interno es el más cercano al almacenamiento físico, o sea, es el relacionado con la forma en que los datos están realmente almacenados. **El nivel externo** es el más cercano a los usuarios, o sea, es el relacionado con la forma en que los datos son vistos por cada usuario individualmente. **El nivel lógico global** es un nivel intermedio entre los dos anteriores.

Existirán varias "vistas externas" diferentes, siendo cada una representación más o menos abstracta de alguna porción de la BD total y existirá únicamente una "vista general", consistente en una representación también abstracta de la BD en su totalidad. Igualmente, existirá una única "vista interna" que representa a la BD completa, tal y como está realmente almacenada.

Descripción de la Arquitectura de Zope.

La siguiente figura (Figura 2) muestra los componentes que conforman la arquitectura de Zope y las vinculaciones establecidas entre ellos tales como los clientes de protocolos, las bases de datos, los sistemas de archivos y los servidores web.

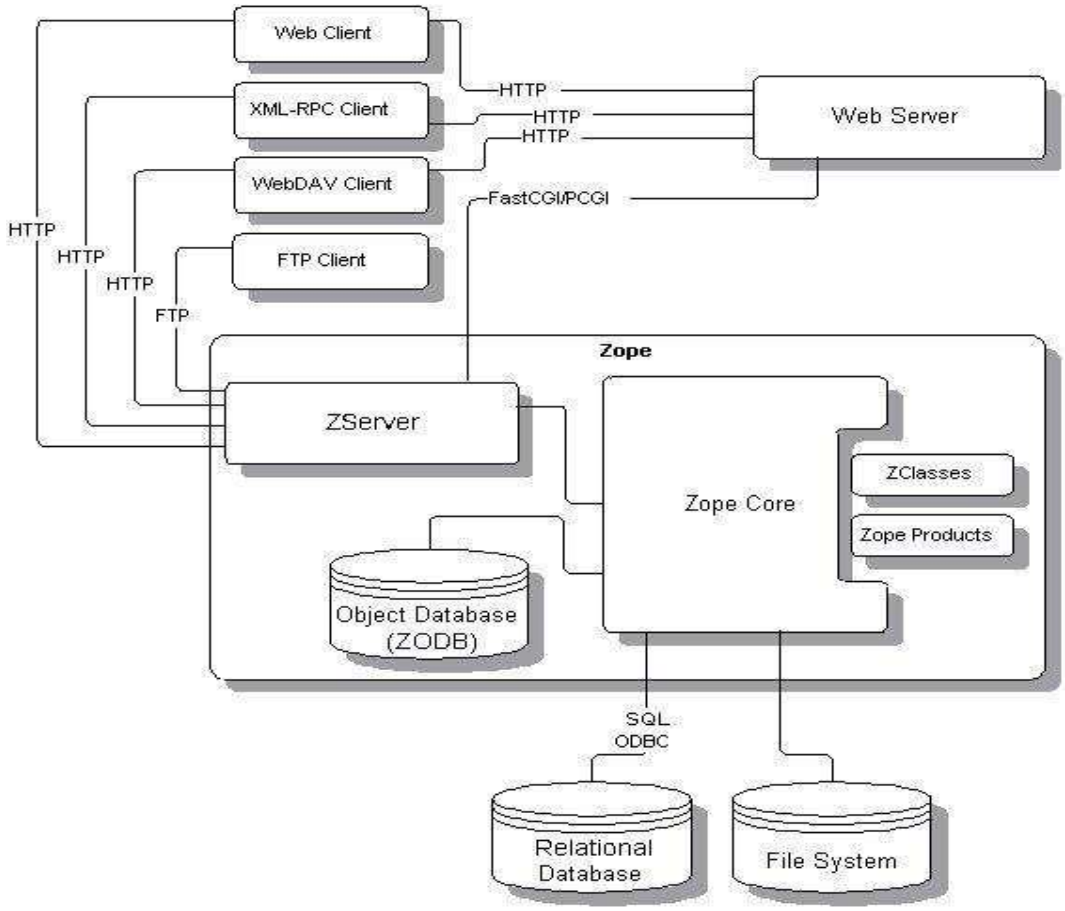

Figura 2. Arquitectura de Zope.

El servidor web incorporado a Zope es el denominado ZServer el cual brinda soporte a clientes HTTP, FTP, XMLRPC y WebDAV, sin embargo, no es obligatorio hacer uso del mismo. Como se puede apreciar, Zope puede interactuar con otros servidores web como Apache o Microsoft IIS (Internet Information Server) haciendo uso de la especificación CGI.

El núcleo de Zope (Zope Core) es la maquinaria que coordina todo el trabajo (la visualización de contenido, el manejo de la interfaz de gestión, el acceso a los objetos de Zope, etc.). Los objetos de Zope se encuentran en una base de datos denominada ZODB (Zope Object Database) aunque no es obligatorio almacenar la información en la misma. Zope es capaz de interactuar con un conjunto amplio de bases de datos relacionales y objeto-relacionales (por ejemplo, Oracle, MySQL, PostgreSQL, Sybase, Interbase, DB2, Informix y Gadfly) y también con el sistema de archivos del sistema operativo del servidor. Los programadores pueden incorporar nuevos tipos de objetos (ZClasses) haciendo uso de la interfaz de gestión de Zope (Zope Management Interface o ZMI) o mediante la incorporación de archivos de productos (Zope Products) al sistema de archivos del servidor Zope. Los productos Zope son programados en Python.

Persistencia de objetos nativos y transacciones.

Los objetos de Zope se encuentran almacenados en una base de datos transaccional orientada a objetos de alto rendimiento conocida como “Zope Object Database (ZODB)". Cada petición web es tratada como una transacción aparte por la base de datos. Si algún error ocurre en la aplicación durante la petición, cualquier cambio realizado durante la petición será automáticamente deshecho. La base de datos permite deshacer en varios niveles, permitiendo al administrador deshacer cambios con solo realizar click sobre el botón correspondiente. El entorno de trabajo de Zope permite que todo lo concerniente a la persistencia y transacciones sea completamente transparente para el desarrollador. Las bases de datos relacionales utilizadas en Zope pueden también moverse en el "Zope's Transaction Framework".

Extensibilidad.

Zope es altamente extensible y usuarios avanzados pueden crear nuevos tipos de objetos Zope, cualquiera puede escribir nuevos complementos para Zope en Python o construyéndolos completamente por medio de la web. El software de Zope provee un número de útiles componentes preconstruidos para ayudar a los desarrolladores, incluyendo una robusta colección de clases que toman en cuenta los detalles de implementar nuevos objetos Zope.

Una cantidad de productos complementarios de Zope que están disponibles proveen cualidades como arrastrar tópicos de discusión, datos de publicación de escritorio, herramientas XML y de implementación de comercio electrónico. Muchos de estos productos han sido escritos por los miembros altamente activos de la comunidad Zope y la mayoría son también de código libre.

Seguridad y Delegación de seguridad.

Una de las cosas que diferencia a Zope de otros servidores de aplicaciones es que este fue diseñado desde el comienzo para ajustarse no solo con los objetos web, sino también con el modelo de desarrollo web. Actualmente las mejores aplicaciones web requieren la participación de varias personas a través de una organización que tiene diferentes áreas de conocimiento. Zope está específicamente diseñado para acomodarse a este modelo, permitiendo a los administradores del sitio delegar de una forma segura el control a los expertos en diseño, en las bases de datos y en el administrador de contenidos.

Un buen sitio web requiere la colaboración de muchas personas de la organización, desarrolladores de aplicaciones, expertos en SQL, administradores de contenidos y frecuentemente de los usuarios finales. En un sitio web convencional, el mantenimiento y la seguridad pueden rápidamente convertirse en un problema. ¿Cuánto control debe dársele al administrador de contenidos? ¿Cómo no darle un login al administrador de contenidos que afecte su seguridad? ¿Qué acerca del código SQL embebido en un archivo ASP en el cual se esta trabajando código que probablemente exponga el login de su base de datos?

Los objetos en Zope proveen un conjunto de permisos mucho más complejos que los sistemas convencionales basados en archivos. Estos permisos varían con cada objeto, dependiendo de las capacidades del objeto. Esto permite la implementación bien lograda de un control de accesos. Por ejemplo, se puede hacer un control de accesos en el cual los administradores de contenidos puedan utilizar objetos "SQL Method", pero que no puedan ver ni cambiar su contenido. También se puede realizar restricciones en las cuales un usuario pueda crear cierta clase de objetos, por ejemplo "Folders" y "DTML Documents" pero no "SQL Methods" u otros objetos.

Zope provee la capacidad de administrar usuarios a través de la web vía "User Folders", los cuales son carpetas especiales que contiene la información del usuario. Muchos complementos de Zope están disponibles los cuales permiten extender los tipos de "User Folders" que contienen la información de los usuarios de fuentes externas como bases de datos relacionales o directorios LDAP. La capacidad de agregar "User Folders" puede ser delegada a usuarios dentro de una subcarpeta, esencialmente permitiendo delegar la creación y la administración de subsecciones de un sitio web a usuarios poco confiables sin preocuparse de que los usuarios cambien los demás objetos aparte de los de su carpeta.

Requisitos no Funcionales de la Base de Datos.

1. La base de datos deberá ser independiente de la aplicación.
2. El servidor será montado en una P4 con procesador de 256 MB de RAM como mínimo.
3. La BD deberá tener una gran disponibilidad y confiabilidad en cuánto a los datos que almacena.
4. Se deberá reducir el tiempo de respuesta a las peticiones por parte de los usuarios a la BD.
5. Se deberá trazar una estrategia de mantenimiento de la BD como norma de soporte.
6. La información manejada por la BD debe estar protegida de acceso no autorizado y divulgación.
7. La información manejada por la BD será objeto de cuidadosa protección contra la corrupción y estados inconsistentes.
8. La BD deberá cumplir con el siguiente estándar de codificación para llevar una buena organización.
9. Se empleará el siguiente estándar de codificación:

Sintaxis para los identificadores:

- Se componen de siglas y palabras del español en minúscula y separadas por subrayado "_".
- Tienen que resultar lo más resumido posible y comprensibles.
- Esquemas: e_nombreesquema
- Bases de Datos: bd_nombrebasedatos
- Tablas: t_nombretabla
- Columnas: c_nombrecolumna
- Checks: ch_numerocheck
- Funciones: f_nombrefuncion
- Parámetros: p_nombreparametro
- Variables: v_nombrevariable
- Vistas: vi_nombrevista
- Triggers: tr_nombretrigger

Estrategia de Integración de la BD con Zope.

PostgreSQL constituye uno de los SGBD con los cuales Zope puede interactuar. La conexión desde Zope hacia un SGBD se puede realizar de dos formas:

1. Utilizando un adaptador (Zope Database Adapter o ZDA) el cual permite establecer las conexiones entre Zope y el SGBD. Esta opción es útil cuando se quiere gestionar la conexión visualmente en el ZMI y las operaciones sobre la BD se realizan a través de los objetos de Zope denominados "Métodos ZSQL" (Z SQL Methods) los cuales se apoyan en algunas etiquetas DTML. Los adaptadores de BD son importados, agregados y utilizados en el Zope como cualquier otro producto.

2. Utilizando un driver de Python para el acceso al SGBD. Esta opción es útil cuando se programan scripts Python internos o externos a Zope que se conectan a la BD. El módulo tiene que ser instalado en el Python que reside en el framework de Zope.

Se propone el diseño de un producto Zope que utilice un driver Python para acceder a la BD. El módulo psycopg2 es utilizado en Python para acceder a PostgreSQL el cual cumple con la especificación de Python DBAPI 2.0 (Database API Specification v2.0), es bastante robusto, hace una eficiente gestión de las conexiones mediante pooling y actualmente es objeto de continuas mejoras.

Para instalar el módulo psycopg2 se utiliza el comando **python setup.py install** en el shell del Python de Zope. Esto ubica los ficheros en la dirección Python23/Lib/site-packages dentro de la carpeta de instalación de Zope.

Con el psycopg2 instalado se puede programar el acceso a los datos de la aplicación y los módulos Python que se obtengan se copian en una carpeta ubicada en la dirección Zope-Instance/Products a la cual se le debe incluir un script con nombre __init__.py que contenga las instrucciones de seguridad para esos módulos.

Zope presenta un esquema de seguridad potente y para ejecutar productos hechos por terceros es necesario trabajar con lo que se conoce como *instrucciones de seguridad* con el objetivo de que Zope permita la ejecución del producto sin restricción.

Diseño de la Base de Datos.

El diseño de la base de datos está dividido en cuatro módulos: Reportes de indisciplinas, Medios Básicos, Alojamiento y Seguridad. A continuación se realiza la descripción de dichos módulos.

Módulo Reportes de Indisciplinas.

Este módulo almacena toda la información referente a los reportes de indisciplinas en la Residencia de la UCI y las sanciones aplicadas. Los reportes de indisciplinas están formados por un título, la fecha de creado el reporte, la fecha de cometida la indisciplina, el lugar donde se cometió la indisciplina, si existen trabajadores implicados, una descripción de la indisciplina, las facultades de los estudiantes implicados, el nombre de la persona que reporta, su cargo y área a la que pertenece. Cuando se emite un reporte las comisiones disciplinarias sancionan a los estudiantes de cada facultad que están implicados en la indisciplina. Cada estudiante es sancionado solo con un tipo de sanción. De los estudiantes sancionados de cada facultad solo se registra la cantidad por tipo de sanción.

De acuerdo a lo anterior se propone el siguiente modelo lógico de la base de datos:

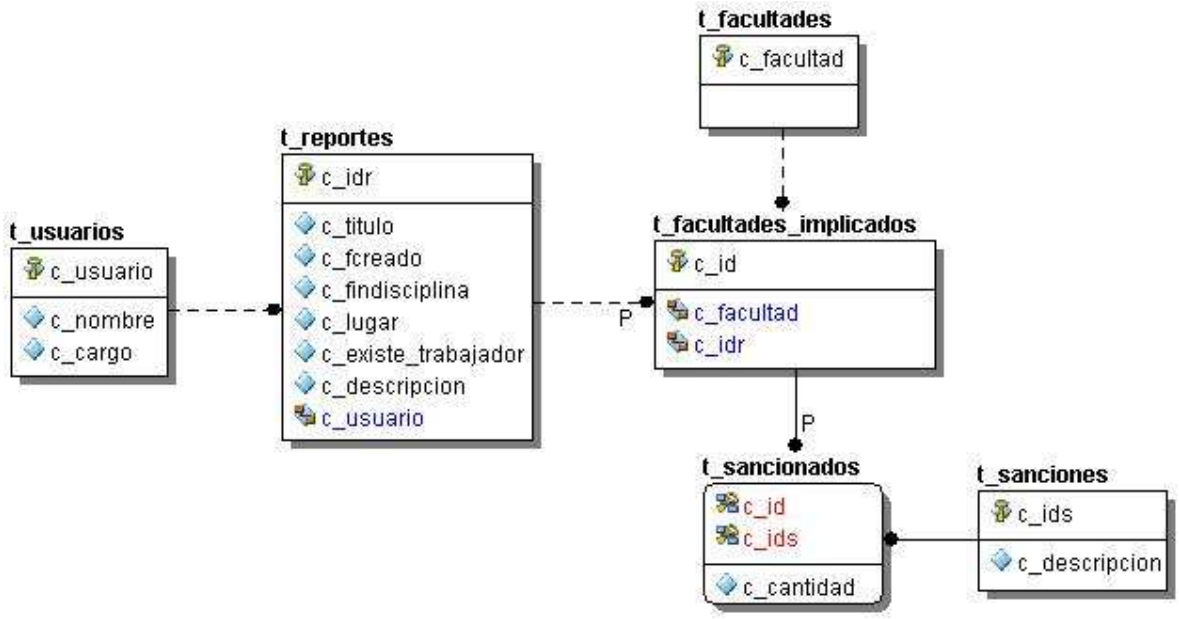

Figura 3. Modelo lógico para el módulo Reportes de Indisciplinas.

Descripción de las interrelaciones.

Las interrelaciones del modelo anterior obedecen a que un usuario puede confeccionar muchos reportes de indisciplinas y cada reporte es confeccionado por un solo usuario. Un reporte tiene estudiantes implicados de una o más facultades y una facultad puede tener estudiantes implicados en muchos reportes. De cada facultad con estudiantes implicados en un reporte se sancionan a los estudiantes por tipo de sanción y cada tipo de sanción puede ser aplicado a muchos estudiantes sancionados.

El modelo físico que se obtiene es el siguiente:

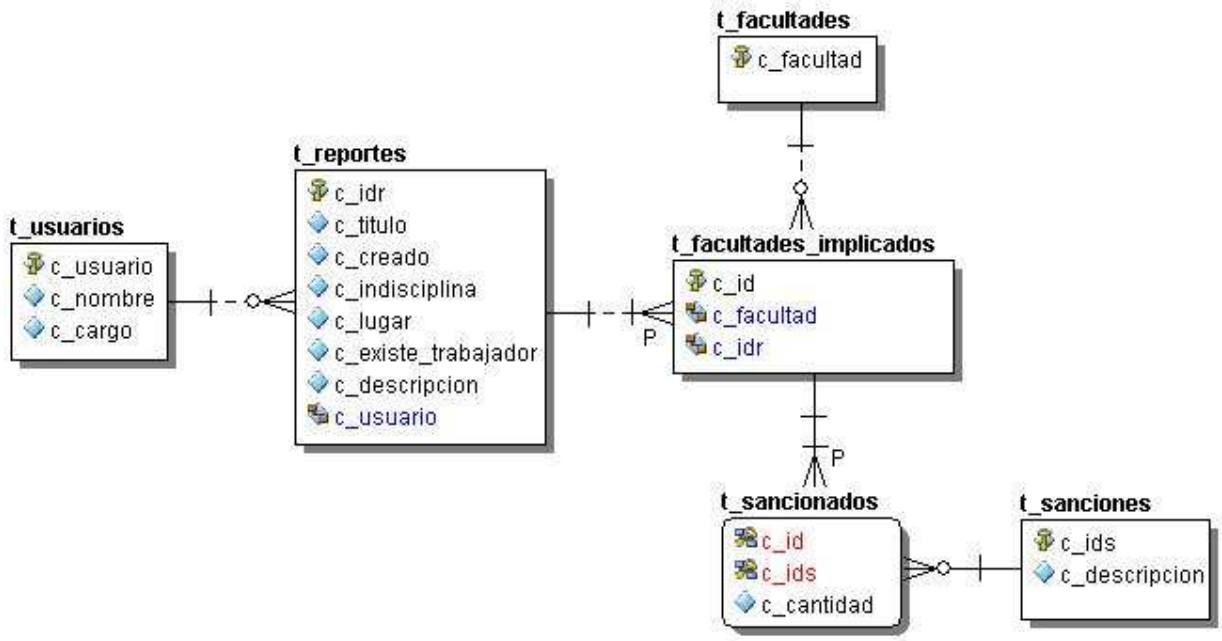

Figura 4. Modelo físico para el módulo Reportes de Indisciplinas.

Descripción de las Tablas.

Nombre	t_facultades			
Descripción	Almacena los números de las facultades de la UCI.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_facultad	char(2)	PK	Número de facultad.

Nombre	t_reportes			
Descripción	Almacena la información de todos los reportes de indisciplinas.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_idr	serial	PK	Valor entero auto-numérico.
	c_titulo	varchar(50)	Not Null	Título del reporte.
	c_fcreado	date	Not Null	Fecha de creado.
	c_findisciplina	date	Not Null	Fecha en que ocurrió la indisciplina.
	c_lugar	varchar(100)	Not Null	Lugar donde ocurrió la indisciplina.
	c_existe_trabajador	boolean	Not Null	'true' si existen trabajadores involucrados. 'false' en caso contrario.
	c_descripcion	text	Not Null	Descripción de la indisciplina.
FK	c_ usuario	varchar(20)	Not Null	Llave foránea de "t_usuarios".

Nombre	t_ facultades_implicados			
Descripción	Almacena las referencias de las facultades con estudiantes implicados en cada reporte.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_id	serial	PK	Valor entero auto-numérico.
FK	c_ facultad	char(2)	Not Null	Llave foránea de "t_facultades".
FK	c_idr	integer	Not Null	Llave foránea de "t_reportes".

Nombre	t_ sancionados			
Descripción	Almacena la cantidad de estudiantes sancionados de cada facultad por reporte.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_id	integer	PK	Llave primaria y foránea de "t_ facultades_implicados".
PKF	c_ ids	integer	PK	Llave primaria y foránea de la tabla "t_ sanciones".
	c_cantidad	integer	Not Null	Cantidad de sancionados.

Nombre	t_ sanciones			
Descripción	Almacena los tipos de sanción.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_ ids	serial	PK	Valor entero auto-numérico.
	c_descripcion	text	Not Null	Descripción del tipo de sanción.

Descripción de las Funciones Almacenadas.

Función	f_insertar_reporte()	
Descripción	Inserta un reporte de indisciplina con los valores de los parámetros de entrada.	
Parámetros	Tipo	Descripción
v_titulo	varchar	Titulo del reporte.
v_fcreado	date	Fecha de creado el reporte.
v_findisciplina	date	Fecha en que ocurre la indisciplina.
v_lugar	varchar	Lugar donde ocurre la indisciplina.
v_existe_trabajador	boolean	Descripción de la indisciplina.
v_descripcion	text	Login del usuario que emite el reporte.
v_usuario	varchar	Valor booleano por si existe o no trabajadores involucrados en la indisciplina.
Resultado	integer	Id. del reporte recién insertado.

Función	f_actualizar_reporte()	
Descripción	Actualiza un reporte de indisciplina con los valores de los parámetros de entrada.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
v_titulo	varchar	Titulo del reporte.
v_ fcreado	date	Fecha de creado el reporte.
v_ findisciplina	date	Fecha en que ocurre la indisciplina.
v_lugar	varchar	Lugar donde ocurre la indisciplina.
v_existe_trabajador	boolean	Valor booleano por si existe o no trabajadores involucrados en la indisciplina.
v_descripcion	text	Descripción de la indisciplina.
v_usuario	varchar	Login del usuario que emite el reporte.
Resultado	void	

Función	f_eliminar_reporte()	
Descripción	Elimina todos los datos de un reporte dado su Id.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
Resultado	void	

Función	f_cargar_sancionados()	
Descripción	Devuelve todos los sancionados de una facultad implicados en un reporte con el Id. del tipo de sanción correspondiente.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
v_facultad	char	Número de la facultad.
Resultado	Set of record	Sancionados de la facultad implicados el reporte con el Id. del tipo de sanción correspondiente.

Función	f_cargar_reporte()	
Descripción	Devuelve datos de un reporte dado su Id.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
Resultado	record	Record formado por los siguientes datos del reporte: - Titulo. - Fecha de creado. - Fecha de la indisciplina. - Lugar donde ocurrió la indisciplina. - Si existe o no trabajadores involucrados. - Descripción de la indisciplina. - Nombre de la persona que emitió el reporte. - Cargo de la persona que emitió el reporte.

Función	f_insertar_fac_reporte()	
Descripción	Esta función inserta una facultad con estudiantes implicados en un reporte de indisciplina.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
v_facultad	char	Número de la facultad.
Resultado	void	

Función	f_cargar_facs_reporte()	
Descripción	Devuelve todas las facultades con estudiantes involucrados en un reporte.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
Resultado	Set of char	Números de las facultades con estudiantes involucrados en el reporte.

Función	f_eliminar_facs_reporte()	
Descripción	Elimina todas las facultades y los estudiantes implicados en un reporte.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
Resultado	void	

Función	f_eliminar_sancionados()	
Descripción	Elimina todos los sancionados de una facultad que estén involucrados en un reporte determinado.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
v_facultad	char	Número de la facultad.
Resultado	void	

Función	f_insertar_sancionados()	
Descripción	Inserta una cantidad de estudiantes sancionados de una facultad involucrados en un reporte.	
Parámetros	Tipo	Descripción
v_idr	integer	Id. del reporte.
v_facultad	char	Número de la facultad.
v_ids	integer	Id. del tipo de sanción.
v_cantidad	integer	Cantidad de estudiantes sancionados.
Resultado	void	

Módulo Medios Básicos.

Este módulo almacena la información de todos los medios básicos que se encuentran en el área de la Residencia de la UCI. De cada medio se conoce su descripción, precio, fecha de alta (es la fecha de entrada de un medio a un local), descripción de la avería (si tiene) y local donde pertenece. Los medios son Activos Fijos Tangibles (AFT) o Útiles. De cada Útil se conoce cuántos están bien, cuántos están regular y cuántos están mal. De los AFT se conoce su número de inventario (es único para cada AFT) y su estado (bien, regular, mal). Los AFT son Muebles o Equipos. De los Muebles no se tiene información adicional. En el caso de los Equipos se conoce además el número de serie, la marca y el modelo. También se conoce la cantidad de medios de un tipo que debe haber en un local (valor conocido como disposición). Los medios pueden ser trasladados de un local a otro con carácter permanente o temporal. Los medios inventariados están en apartamentos, talleres, almacenes, sub-nodos, cuartos de desahogo y cuartos eléctricos. Excepto los apartamentos, solo interesa guardar los nombres de dichos locales. De los cuartos eléctricos se conoce el edificio al que pertenece. De cada edificio se conoce su número al igual que los apartamentos.

De acuerdo a lo anterior se propone el siguiente modelo lógico de la base de datos:

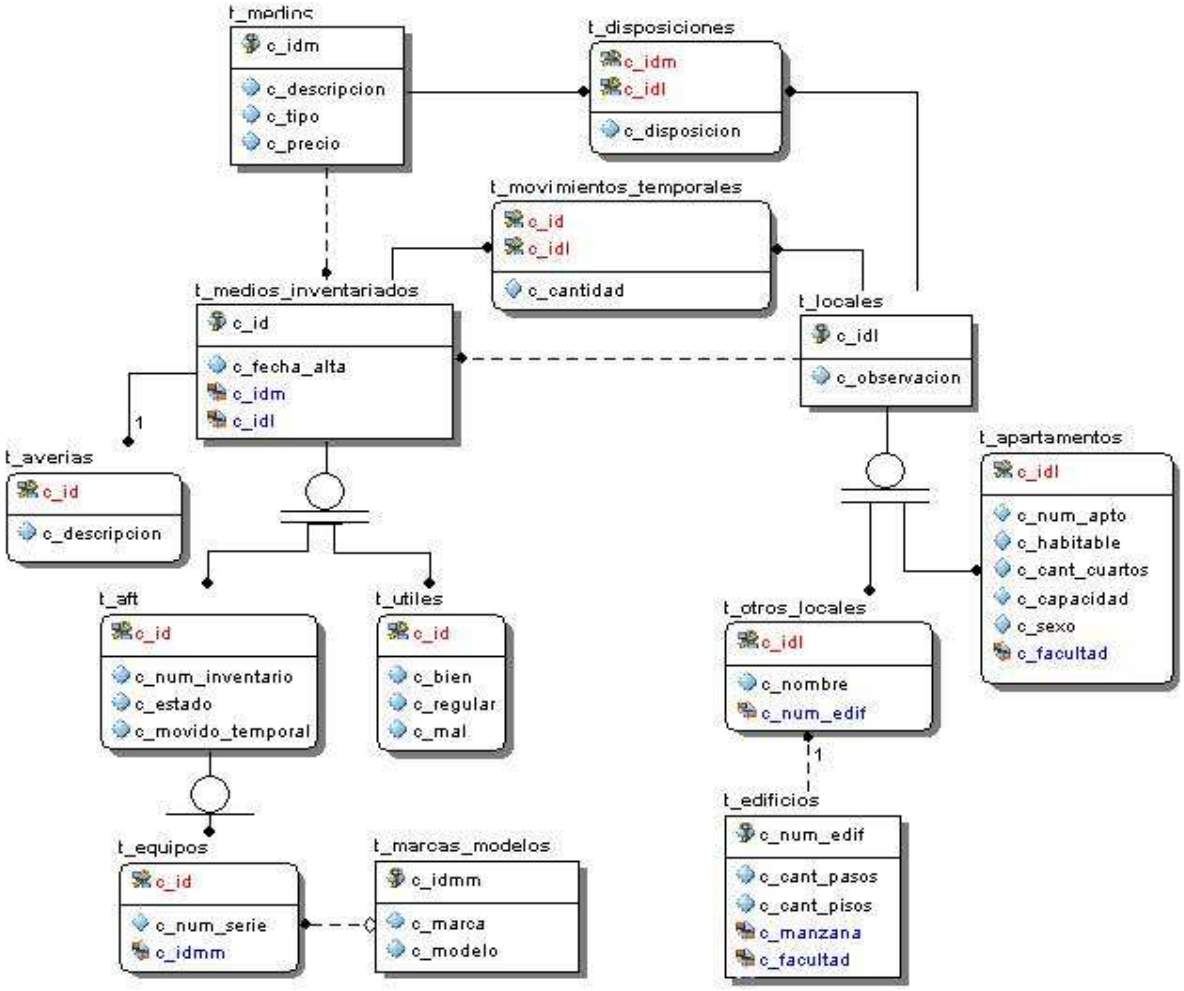

Figura 5. Modelo lógico para el módulo Medios Básicos.

Descripción de las interrelaciones.

Las interrelaciones del modelo anterior obedecen a que de cada tipo de medio pueden existir muchos medios inventariados y cada medio inventariado es de un tipo. Cada tipo de medio puede tener una disposición en muchos locales y un local puede tener disposiciones de muchos tipos de medios. Los medios inventariados pueden tener una avería y cada avería es de un solo medio inventariado. Los medios inventariados están en un local y cada local puede tener muchos medios inventariados. Los medios inventariados pueden estar movidos temporalmente hacia otro local y en un local pueden existir muchos medios con carácter temporal. Los medios inventariados son AFT o útiles.

Un AFT puede ser un equipo u otro tipo. Cada equipo es de una marca y modelo y muchos equipos pueden ser de la misma marca y modelo. Un local puede ser un apartamento o cualquier otro de los locales. Un cuarto eléctrico está ubicado en un edificio y un edificio puede tener un cuarto eléctrico. Cada apartamento pertenece a un solo edificio y cada edificio tiene muchos apartamentos.

El modelo físico que se obtiene es el siguiente:

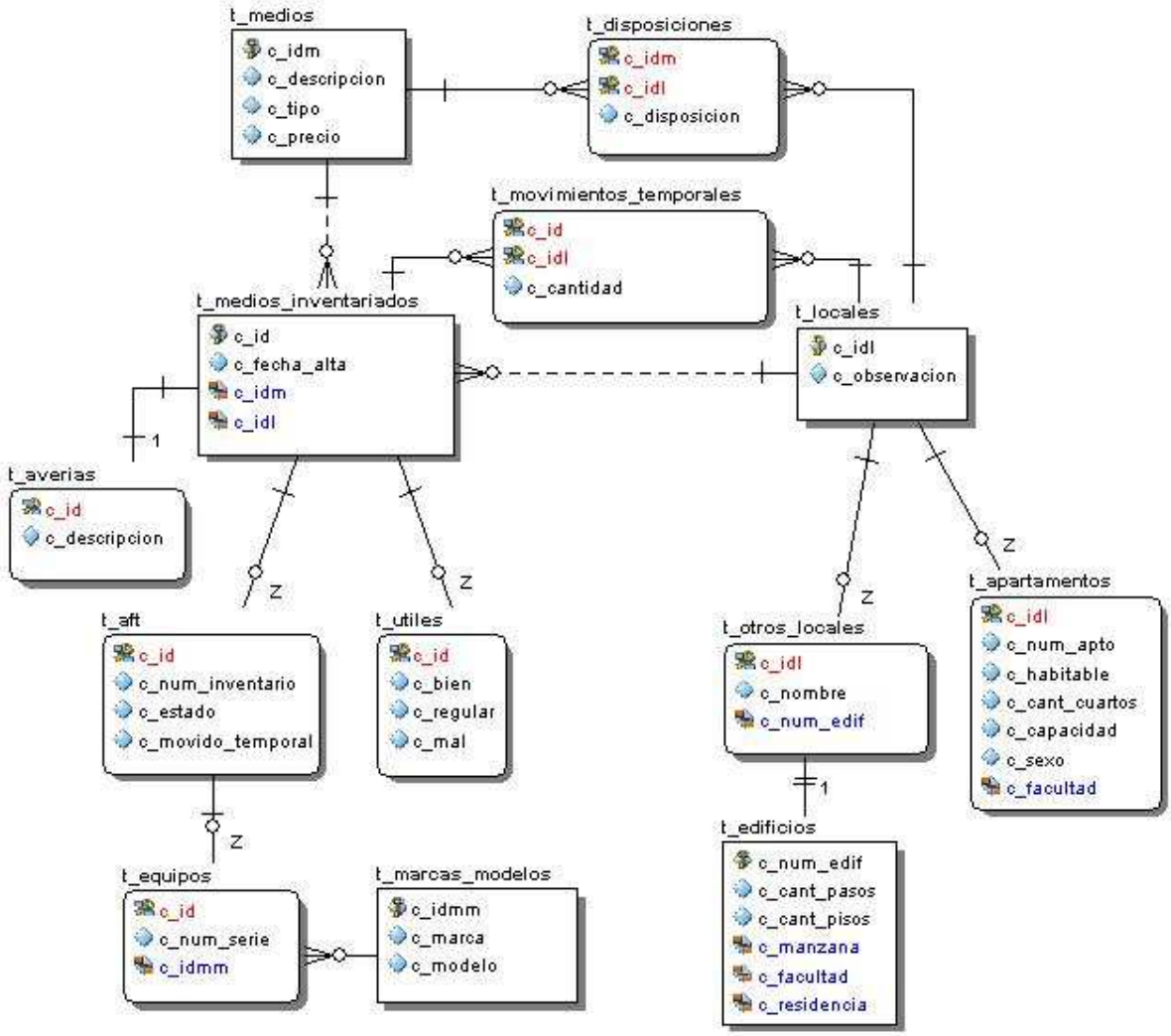

Figura 6. Modelo físico para el módulo Medios Básicos.

Descripción de las Tablas.

Nombre	t_medios			
Descripción	Almacena la información de todos los tipos de medios.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_idm	serial	PK	Valor entero auto-numérico.
	c_descripcion	text	Not Null	Descripción del tipo de medio
	c_tipo	char(1)	Not Null	Tipo de medio: - 'm' si es un Mueble - 'e' si es un Equipo - 'u' si es un Útil
	c_precio	money	Null	Precio del tipo de medio.

Nombre	t_medios_inventariados			
Descripción	Almacena información de todos los medios inventariados.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_id	serial	PK	Valor entero auto-numérico.
	c_fecha_alta	date	Not Null	Fecha de entrada del medio a un local.
FK	c_idm	integer	Not Null	Llave foránea de la tabla "t_medios".
FK	c_idl	integer	Null	Llave foránea de la tabla "t_locales".

Nombre	t_averias			
Descripción	Almacena la información de todos los reportes de medios averiados.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_id	integer	PK	Llave primaria y foránea de la tabla "t_medios_inventariados".
	c_descripción	text	Not Null	Descripción de la avería.

Nombre	t_utiles			
Descripción	Almacena información de todos los útiles.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_id	integer	PK	Llave primaria y foránea de la tabla "t_medios_inventariados".
	c_bien	integer	Null	Cantidad de medios de estado 'bien' en un local.
	c_regular	integer	Null	Cantidad de medios de estado 'regular' en un local.
	c_mal	integer	Null	Cantidad de medios de estado 'mal' en un local.

Nombre	t_locales			
Descripción	Almacena información de todos los locales.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_idl	serial	PK	Valor entero auto-numérico.
	c_observacion	text	Null	Observación opcional sobre el local.

Nombre	t_aft			
Descripción	Almacena información de todos los activos fijos tangibles (AFT).			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_id	integer	PK	Llave primaria y foránea de la tabla "t_medios_inventariados".
	c_num_inventario	varchar(15)	Null	Número de inventario del AFT.
	c_estado	char(1)	Not Null	Estado del AFT: -'b' si tiene en buen estado -'r' si tiene un estado regular -'m' si está en mal estado
	c_movido_temporal	boolean	Not Null	'true' si el AFT está movido temporalmente, 'false' en caso contrario.

Nombre	t_equipos			
Descripción	Almacena información de todos los equipos.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_id	integer	PK	Llave primaria y foránea de la tabla "t_aft".
	c_num_serie	varchar(10)	Null	Número de serie del equipo.
FK	c_idmm	char(1)	Null	Llave foránea de la tabla "t_marcas_modelos".

Nombre	t_marcas_modelos			
Descripción	Almacena la información de todas las marcas y modelos de los equipos.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_idmm	serial	PK	Valor entero auto-numérico.
	c_marca	varchar(20)	Null	Marca de equipo.
	c_modelo	varchar(20)	Null	Modelo de equipo.

Nombre	t_disposiciones			
Descripción	Almacena la información de todas las disposiciones de medios en un local.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_idm	integer	PK	Llave primaria y foránea de la tabla “t_medios”.
PKF	c_idl	integer	PK	Llave primaria y foránea de la tabla “t_locales”.
	c_disposicion	integer	Not Null	Cantidad de medios de un tipo que debe haber en el local.

Nombre	t_movimientos_temporales			
Descripción	Almacena la información de todos los movimientos temporales de los medios.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_id	integer	PK	Llave primaria y foránea de la tabla “t_medios_inventariados”.
PKF	c_idl	integer	PK	Llave primaria y foránea de la tabla “t_locales”.
	c_cantidad	integer	Not Null	Cantidad de medios que se han movido temporalmente hacia otro local.

Nombre	t_otros_locales			
Descripción	Almacena información de los locales que no son apartamentos.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_idl	integer	PK	Llave primaria y foránea de la tabla “t_locales”.
	c_nombre	varchar(50)	Not Null	Nombre que recibe el local.
FK	c_num_edif	char(3)	Null	Llave foránea de la tabla “t_edificios”.

Módulo Alojamiento.

Este módulo almacena la estructura de la Residencia de la UCI y la ubicación de todos sus residentes. La Residencia de la UCI está compuesta por 4 direcciones de residencia: 3 de estudiantes y 1 de profesores. De cada edificio se necesita almacenar su número, cantidad de pasos, cantidad de pisos, número de la manzana donde está ubicado y dirección de residencia y número de la facultad a las que pertenece. De los apartamentos se conoce su número (incluye el número de edificio), si está habitado, cantidad de cuartos, capacidad máxima, sexo y facultad a la que pertenece. En el caso de los residentes se conoce su nombre y apellidos, municipio, provincia, dirección de correo, número del apartamento donde está ubicado, número de teléfono y estado. El estado del residente puede ser normal, licencia o baja. En los casos de licencia y baja el residente pierde la ubicación en su apartamento. Se necesita mantener un registro de todas las entradas y salidas de los residentes en los apartamentos. Existen datos de los residentes que se proveen a la aplicación por medio de servicios web tal es el caso del nombre, apellidos, municipio, provincia, dirección de correo y el número de teléfono del apartamento.

De acuerdo a lo anterior se propone el siguiente modelo lógico de la base de datos:

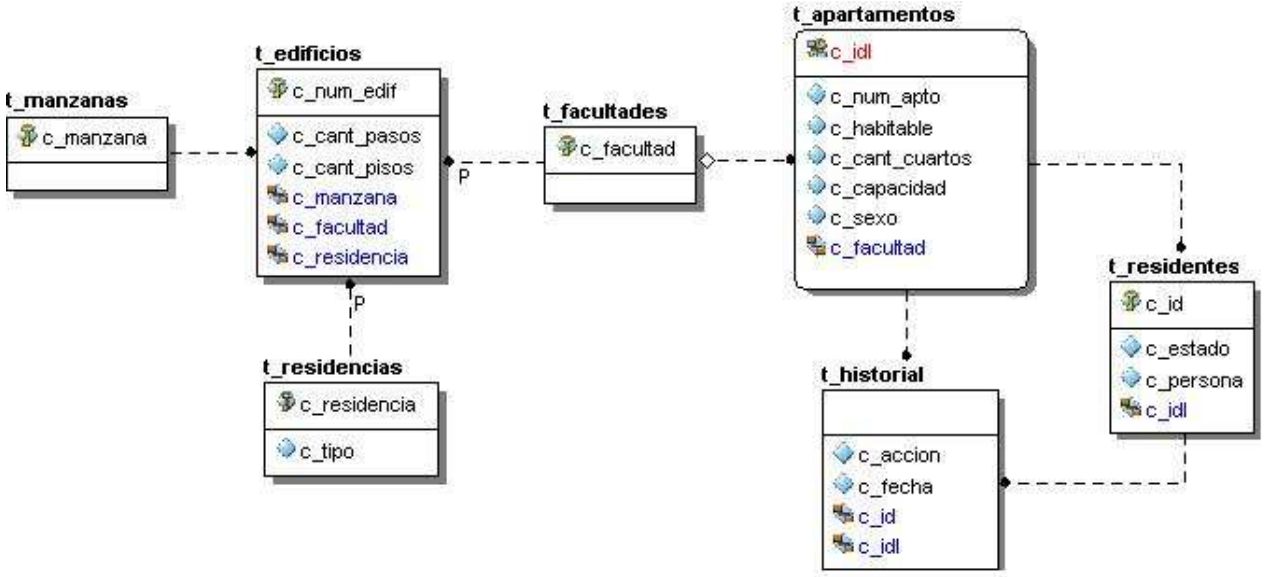

Figura 7. Modelo lógico para el módulo Alojamiento.

Descripción de las interrelaciones.

Las interrelaciones del modelo anterior obedecen a que en una manzana pueden existir muchos edificios y un edificio está ubicado en una manzana. Cada edificio es ocupado por una dirección de residencia y cada dirección de residencia ocupa muchos edificios. Cada edificio pertenece a una facultad y a cada facultad le pertenecen varios edificios. Cada apartamento está asignado a una facultad y a cada facultad le pertenecen muchos apartamentos. Cada apartamento puede estar habitado por varios residentes y cada residente está ubicado en un apartamento. Un registro del historial hace referencia a una acción de entrada/salida en un apartamento y un apartamento puede tener muchos registros de entrada/salida. Un registro del historial hace referencia a una acción de entrada/salida de un residente en un apartamento y un residente puede tener muchos registros de entrada/salida.

El modelo físico que se obtiene es el siguiente:

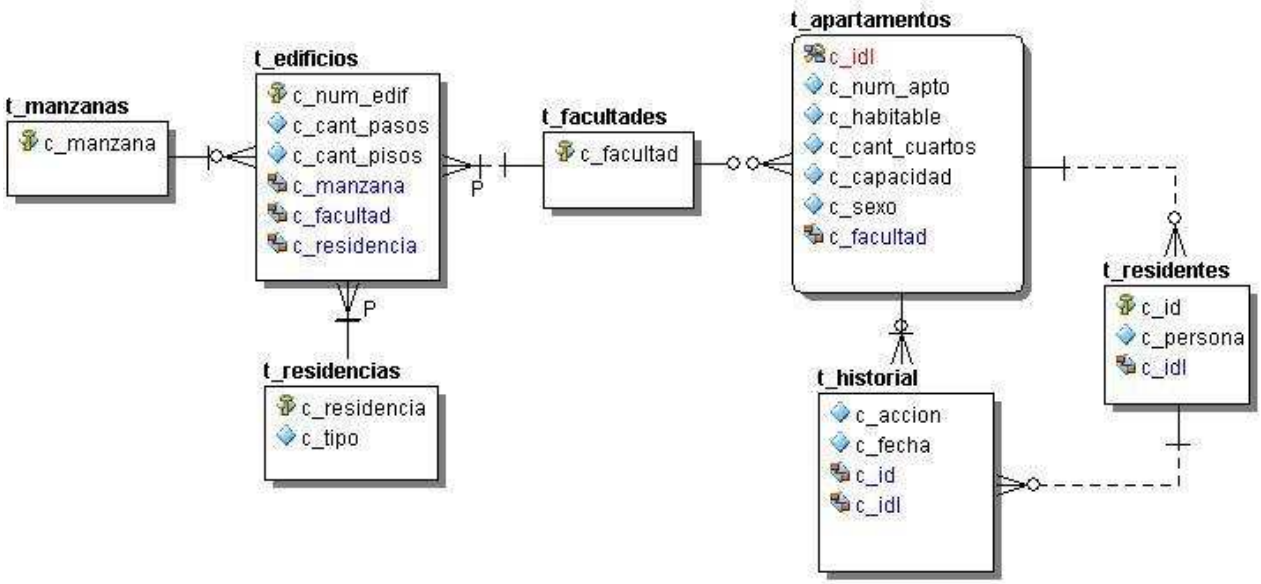

Figura 8. Modelo físico para el módulo Alojamiento.

Descripción de las Tablas.

Nombre	t_apartamentos			
Descripción	Almacena la información de todos los apartamentos.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_idl	integer	PK	Llave primaria y foránea de la tabla "t_locales".
	c_num_apto	char(6)	Not Null	Número del apartamento.
	c_habitable	boolean	Not Null	'true' si está habitado por estudiantes o trabajadores, 'false' en caso contrario.
	c_cant_cuartos	integer	Not Null	Cantidad de cuartos del apartamento.
	c_capacidad	integer	Not Null	Capacidad máxima del apartamento.
	c_sexo	char(1)	Null	Sexo de las personas del apartamento.
FK	c_facultad	char(2)	Null	Llave foránea de "t_facultades".

Nombre	t_manzanas			
Descripción	Almacena los números de todas las manzanas.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_manzana	char(2)	PK	Número de la manzana.

Nombre	t_edificios			
Descripción	Almacena la información de todos los edificios.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_num_edif	char(3)	PK	Número del edificio.
	c_cant_pasos	integer	Not Null	Cantidad de pasos del edificio.
	c_cant_pisos	integer	Not Null	Cantidad de pisos del edificio.
FK	c_manzana	char(2)	Not Null	Llave foránea de "t_manzanas".
FK	c_facultad	char(2)	Not Null	Llave foránea de "t_facultades".
FK	c_residencia	char(1)	Not Null	Llave foránea de "t_residencias".

Nombre	t_residencias			
Descripción	Almacena la información de todas las direcciones de residencia.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_residencia	char(1)	PK	Número de la dirección de residencia.
	c_tipo	char(1)	Not Null	Tipo de dirección de residencia: - 'e' si es de estudiantes - 'p' si es de profesores

Nombre	t_residentes			
Descripción	Almacena el login de dominio de todos los residentes.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_id	serial	PK	Valor entero auto-numérico.
	c_persona	varchar(20)	Not Null	Login de dominio del residente.
	c_estado	char(1)	Not Null	Estado del residente: - ‘n’ si es normal - ‘l’ si es licencia - ‘b’ si es baja
FK	c_idl	integer	Not Null	Llave foránea de “t_apartamentos”.

Nombre	t_historial			
Descripción	Registra todas las entradas y salidas de residentes en los apartamentos.			
Llave	Columna	Dato	Requerido	Descripción
	c_accion	char(1)	Not Null	Identificador de la acción: - ‘e’ si es una entrada - ‘s’ si es una salida
	c_fecha	date	Not Null	Fecha en que ocurrió la acción.
FK	c_id	integer	Not Null	Llave foránea de la tabla “t_residentes”.
FK	c_idl	integer	Not Null	Llave foránea de “t_apartamentos”.

Módulo Seguridad.

Este módulo almacena los usuarios de la aplicación y sus permisos por módulo. En la aplicación existen tres niveles de usuarios. El usuario de nivel 1 es el súper usuario. El usuario de nivel 2 es el usuario económico de dirección de residencia que tiene acceso solo a la información de la residencia a la que pertenece. El usuario de nivel 3 es el usuario de facultad que tiene acceso solo a la información de la facultad a la que pertenece. Los usuarios se autentifican por el dominio UCI.

De acuerdo a lo anterior se propone el siguiente modelo lógico de la base de datos:

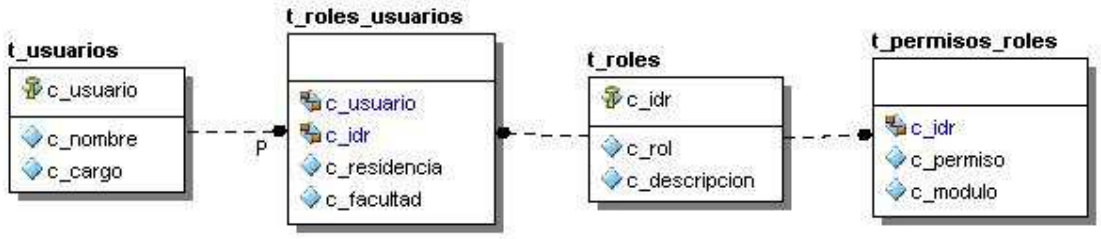

Figura 9. Modelo lógico para el módulo Seguridad.

Descripción de las interrelaciones.

Las interrelaciones del modelo anterior obedecen a que un usuario puede tener varios roles de nivel de residencia o facultad y un rol puede ser de varios usuarios. Un rol puede contener muchos permisos a los módulos de la aplicación.

El modelo físico que se obtiene es el siguiente:

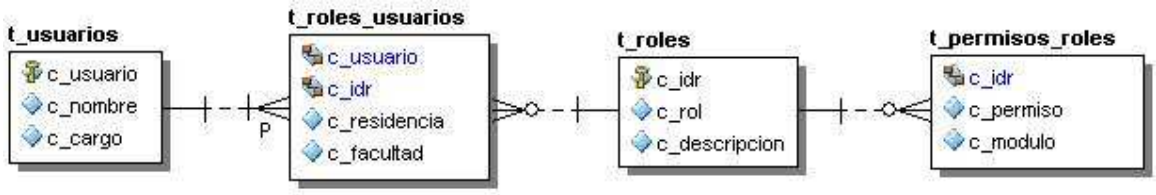

Figura 10. Modelo físico para el módulo Seguridad.

Descripción de las Tablas.

Nombre	t_usuarios			
Descripción	Almacena información de todos los usuarios de la aplicación.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_usuario	varchar(20)	PK	Login de dominio del usuario.
	c_nombre	varchar(50)	Not Null	Nombre y apellidos del usuario.
	c_cargo	varchar(50)	Not Null	Nombre del cargo.

Nombre	t_roles			
Descripción	Almacena la información de todos los roles de la aplicación.			
Llave	Columna	Dato	Requerido	Descripción
PK	c_idr	serial	PK	Valor entero auto-numérico.
	c_rol	varchar(10)	Not Null	Nombre del rol.
	c_descripcion	varchar(50)	Not Null	Descripción del rol.

Nombre	t_roles_usuarios			
Descripción	Almacena informacion sobre los roles de cada usuario.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_usuario	varchar(20)	PK	Llave primaria y foránea de la tabla "t_usuarios".
PKF	c_idr	integer	Not Null	Llave primaria y foránea de la tabla "t_roles".
	c_residencia	integer	Null	Número de residencia donde se aplica el rol.
	c_facultad	integer	Null	Número de la facultad donde se aplica el rol.

Nombre	t_permisos_roles			
Descripción	Almacena informacion sobre los roles de cada usuario.			
Llave	Columna	Dato	Requerido	Descripción
PKF	c_idr	integer	Not Null	Llave primaria y foránea de la tabla "t_roles".
	c_permiso	integer	Not Null	Código del permiso: - '0' para permiso de lectura - '1' para permiso de lectura/escritura
	c_modulo	char(1)	Not Null	Código de módulo: - '1' para módulo "Alojamiento" - '2' para módulo "Medios Básicos"

Acceso a Datos para el Módulo Reportes de Indisciplinas.

Para el acceso a los datos se diseñó un conjunto de clases que garantiza el almacenamiento y recuperación de la información en la base de datos desde una aplicación web. En el diseño se logró una buena extensibilidad, un adecuado desacoplamiento y una óptima distribución de las responsabilidades.

Se emplearon algunos patrones de diseño muy útiles a la hora de implementar el acceso a datos de aplicaciones orientadas a objetos.

Capas: Consiste en descomponer una aplicación en capas ortogonales incrementando los niveles de abstracción desde las capas inferiores hacia las superiores. Este patrón se utiliza en la arquitectura del acceso a datos.

Objeto de Dominio Activo: Consiste en encapsular los detalles del acceso a datos dentro de las implementaciones de las clases persistentes. Este patrón se utiliza en las clases persistentes *ReporteIndisciplina* y *ComisionDisciplinaria*.

Decorador de Recurso: Consiste en añadir comportamientos a un recurso existente con el menor cambio al código cliente. Este patrón se utiliza en las clases dbConnection, pgConnection con el recurso *psycopg2*.

Factoría de Selección: Consiste en generar dinámicamente consultas SQL de selección basado en criterios de búsqueda. Este patrón se utiliza en las clases *FiltroBD* y *Filtro_ReporteIndisciplina*.

Método Factoría: Consiste en centralizar en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la construcción directa del subtipo. Este patrón se utiliza en la clase *ConnectionFactory*.

Arquitectura del Acceso a Datos.

El acceso a los datos de la base de datos está dividido en 3 subsistemas (Figura 11).

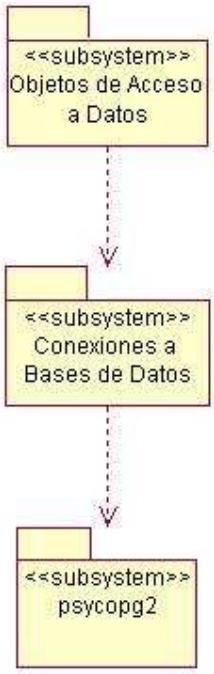

Figura 11. Arquitectura 3 capas del Acceso a Datos.

El subsistema inferior empaqueta las clases del recurso *psycopg2* que gestionan el acceso físico a una base de datos PostgreSQL. El subsistema *Conexiones a Bases de Datos* empaqueta las clases que gestionan las conexiones a cualquier base de datos; en él se encuentran las clases *dbConnection*, *pgConnection* y *ConnectionFactory*. El subsistema *Objetos de Acceso a Datos* empaqueta las clases que son utilizadas por la aplicación para gestionar en sus procesos de negocio la información de una base de datos. En este subsistema se encuentran las clases *ReporteIndisciplina, ComisionDisciplinaria, ListadoBD, FiltroBD y Filtro_ReporteIndisciplina*.

Clases Persistentes.

Las clases persistentes (Figura 12) que se identifican son 2: ReporteIndisciplina y Sanciones. Como se observa en la siguiente figura dichas clases contienen la lógica de acceso a los datos de la base de datos, por lo que desempeñan el rol de objetos activos en el dominio de la aplicación.

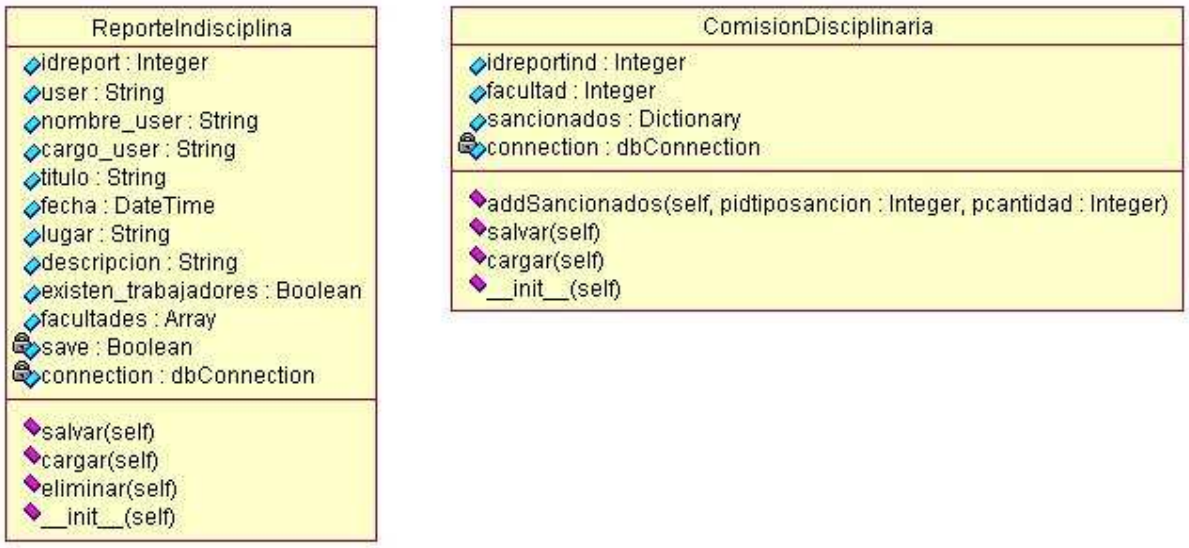

Figura 12. Diagrama de Clases Persistentes.

Descripción de las Clases.

Nombre	ReporteIndisciplina	
Descripción	Gestiona la información de un reporte de indisciplina.	
Atributos	Tipo	Descripción
idreport	Integer	Id. del reporte en la base de datos.
user	String	Nombre de usuario del sistema de la persona que emite el reporte.
nombre_user	String	Nombre y apellidos de la persona que emite el reporte.
cargo_user	String	Cargo de la persona que emite el reporte.
area_user	String	Área a la que pertenece la persona que emite el reporte.
titulo	String	Título del reporte.
fecha	Date	Fecha de ocurrida la indisciplina.
lugar	String	Lugar donde ocurrió la indisciplina.
descripcion	String	Descripción de la indisciplina.
existen_trabajadores	Boolean	Toma True si existen trabajadores involucrados en la indisciplina, False en caso contrario.
facultades	Array	Números de las facultades que tienen estudiantes involucrados en la indisciplina.
save	Boolean	Toma True si el reporte está guardado en la base de datos, False en caso contrario.
connection	dbConnection	Conexión a la base de datos.
Método	salvar()	
Pre-condición	Todos los atributos de la clase tienen que estar instanciados.	
Descripción	Si *save* es True entonces actualiza el reporte en la base de datos. Si *save* es False entonces inserta el reporte en la base de datos y *save* toma valor True.	
Método	cargar()	
Pre-condición	Los atributos *idreporte* y *connection* tienen que estar instanciados.	
Descripción	Instancia los atributos de la clase con la información de un reporte almacenado en la base de datos y *save* toma valor True.	
Método	eliminar()	
Pre-condición	El atributo *idreporte* tiene que estar instanciado.	
Descripción	Elimina el reporte de indisciplina en la base de datos.	
Método	__init__()	
Descripción	Instancia un objeto ReporteIndisciplina con los atributos vacíos y pone el atributo *save* en False.	

Nombre	ComisionDisciplinaria	
Descripción	Gestiona la información de las sanciones aplicadas a los estudiantes de una facultad dado un reporte de indisciplina.	
Atributos	Tipo	Descripción
idreportind	Integer	Id. del reporte en la base de datos.
facultad	Integer	Número de facultad de los estudiantes.
sancionados	Dictionary	Diccionario con ítems de la forma: key → Id. del tipo de sanción value → cantidad de estudiantes sancionados
connection	dbConnection	Objeto conexión a la base de datos.

Método	addSancionados()	
Parámetros	Tipo	Descripción
pidtiposancion	Integer	Id. del tipo de sanción en la base de datos.
pcantidad	Integer	Número de estudiantes sancionados.
Descripción	Adiciona en el diccionario *sancionados* el id. de tipo de sanción y la cantidad de sancionados que se pasan como parámetros.	
Método	salvar()	
Pre-condición	Los atributos *idreportind, facultad* y *connection* tienen que estar instanciados.	
Descripción	Elimina en la base de datos todos los sancionados por tipo de sanción asociados con los atributos *idreportind* y *facultad* e inserta en la base de datos todos los sancionados del diccionario *sancionados*.	
Método	cargar()	
Pre-condición	Los atributos *idreportind, facultad* y *connection* tienen que estar instanciados.	
Descripción	Instancia el atributo *sancionados* con la cantidad de sancionados por tipo de sanción asociados con los atributos *idreportind* y *facultad* que están en la base de datos.	
Método	__init__()	
Descripción	Instancia un objeto ComisionDisciplinaria con los atributos vacíos.	

Clases para la Conexión a la Base de Datos.

Con el propósito de lograr un desacoplamiento de los objetos activos del domino con las conexiones a la base de datos, se diseña una clase abstracta dbConnection (Figura13) que sirve de interfaz para las clases que gestionan las conexiones físicas hacia un gestor determinado. En este caso la clase hija es pgConnection para las conexiones a PostgreSQL. La clase ConnectionFactory se responsabiliza de crear y retornar la conexión adecuada.

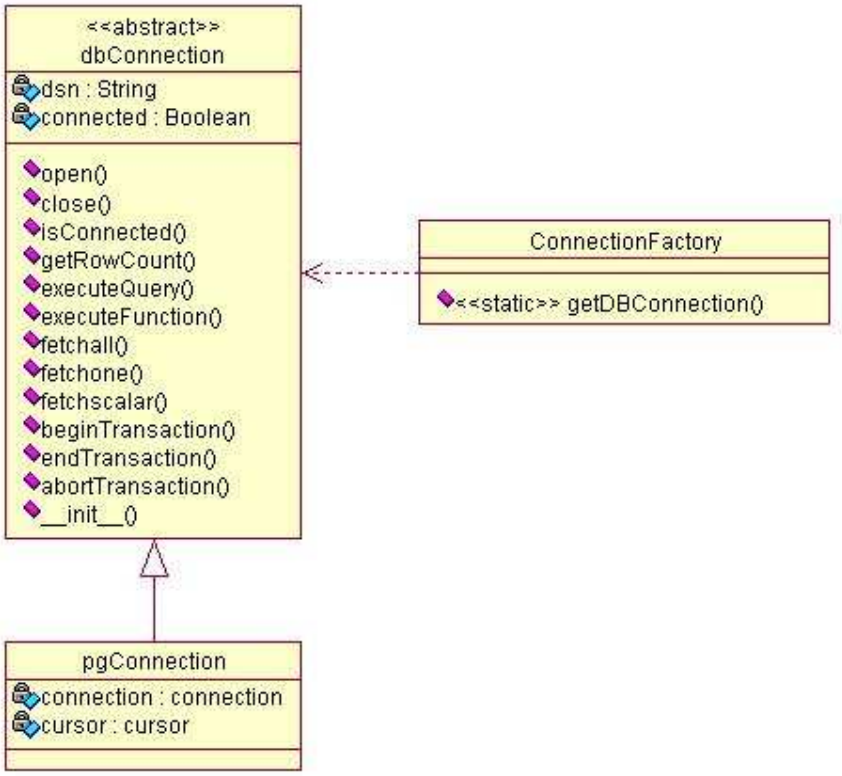

Figura 13. Diagrama de Clases para la Conexión a la BD.

Descripción de las Clases.

Nombre	dbConnection	
Descripción	Clase base para las demás clases que gestionan una conexión hacia una base de datos determinada.	
Atributos	Tipo	Descripción
dsn	String	Cadena de conexión.
connected	Boolean	Estado de la conexión. Toma True si la conexión está abierta, False en caso contrario.

Método	open()	
Pre-condición	El atributo *dsn* tiene que estar instanciado.	
Descripción	Abre la conexión y pone el atributo *connected* en True.	
Método	close()	
Descripción	Cierra la conexión y pone el atributo *connected* en False.	
Método	isConnected()	
Descripción	Devuelve el valor del atributo *connected*.	
Método	getRowCount()	
Pre-condición	El atributo *connected* tiene que estar en True.	
Descripción	Devuelve la cantidad de filas resultantes en la ejecución de una consulta SQL.	
Método	executeQuery()	
Parámetros	Tipo	Descripción
pSQL	String	Consulta SQL.
parameters	Tuple	Tupla con los parámetros de la consulta SQL.
Pre-condición	El atributo *connected* tiene que estar en True.	
Descripción	Ejecuta la consulta SQL que se pasa como parámetro.	
Método	executeFunction()	
Parámetros	Tipo	Descripción
pFName	String	Nombre de la función.
parameters	Tuple	Tupla con los parámetros de la función.
rowType	Dictionary	Diccionario con ítems de la forma: key → identificador de ítem value → tipo de dato
Pre-condición	El atributo *connected* tiene que estar en True.	
Descripción	Ejecuta la función de la base de datos que se pasa como parámetro.	
Método	fetchall()	
Pre-condición	Tiene que haberse ejecutado una consulta SQL.	
Descripción	Devuelve un arreglo de tuplas con el resultado de la últma consulta SQL que se ejecutó.	
Método	fetchone()	
Pre-condición	Tiene que haberse ejecutado una consulta SQL.	
Descripción	Devuelve una tupla con el primer registro del resultado de la últma consulta SQL que se ejecutó.	
Método	fetchscalar()	
Pre-condición	Tiene que haberse ejecutado una consulta SQL.	
Descripción	Devuelve un valor que representa el primer elemento del primer registro del resultado de la últma consulta SQL que se ejecutó.	
Método	beginTransaction()	

Pre-condición	El atributo *connected* tiene que estar en True.	
Descripción	Inicia una transacción de operaciones en la base de datos.	
Método	endTransaction()	
Pre-condición	El atributo *connected* tiene que estar en True. Tiene que existir una transacción de operaciones en la base de datos.	
Descripción	Finaliza la transacción de operaciones en la base de datos.	
Método	endTransaction()	
Pre-condición	El atributo *connected* tiene que estar en True. Tiene que existir una transacción de operaciones en la base de datos.	
Descripción	Aborta una transacción de operaciones en la base de datos.	
Método	__init__()	
Parámetros	Tipo	Descripción
pdsn	String	Cadena de conexión.
Descripción	Instancia un objeto dbConnection con la cadena de conexión que se pasa como parámetro y con el atributo *connected* en False.	

Nombre	ConnectionFactory	
Descripción	Abstrae la construcción de los objetos dbConnection.	
Método	getDBConnection()	
Descripción	Crea y retorna un objeto de tipo dbConnection	

Nombre	pgConnection	
Descripción	Clase que hereda de dbConnection para gestionar una conexión hacia una base de datos PostgreSQL.	
Atributos	Tipo	Descripción
connection	connection	Objeto conexión de la clase connection del paquete psycopg2.
cursor	cursor	Objeto cursor de la clase cursor del paquete psycopg2.

Clases para las Búsquedas en la Base de Datos.

Un aspecto a tener en cuenta en el diseño es la gestión de las búsquedas en la base de datos según los criterios que se generan en tiempo de ejecución. Por dicho motivo se diseña un clase padre FiltroBD (Figura 14) que expone los atributos y métodos necesarios para generar dinámicamente las consultas SQL adecuadas para realizar búsquedas de información. La clase Filtro_ReporteIndisciplina se reponsabiliza de implementar la creación de las consultas para PostgreSQL según los valores que toman sus atributos en los objetos que se crean, los cuales coinciden con los criterios de búsquedas. La clase ListadorBD es una clase que expone métodos que retornan listas de tuplas desde las tablas de la base de datos.

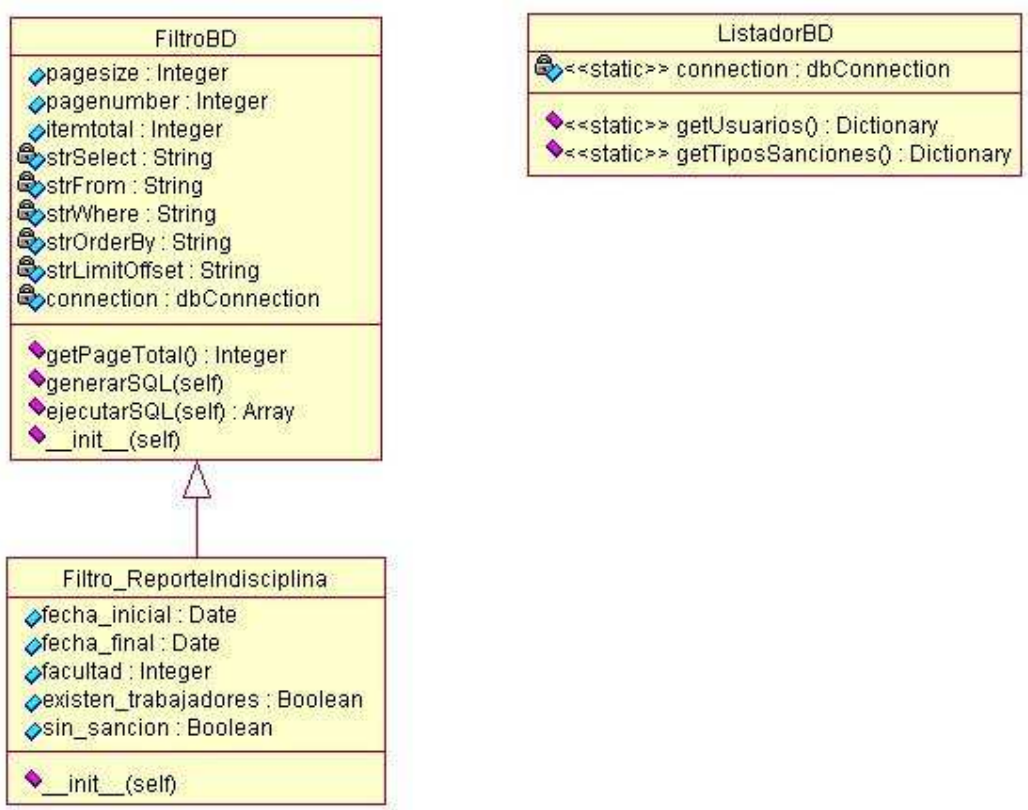

Figura 11. Diagrama de clases para las búsquedas en la BD.

Descripción de las Clases.

Nombre	FiltroBD	
Descripción	Clase base para las demás clases que realizan búsquedas en una base de datos según determinados criterios.	
Atributos	Tipo	Descripción
pagesize	Integer	Tamaño de las páginas de datos.
pagenumber	Integer	Número de una página de datos.
itemtotal	Integer	Número total de ítems en las páginas de datos.
strSelect	String	Cláusula SELECT de una consulta SQL.
strFrom	String	Cláusula FROM de una consulta SQL.
strWhere	String	Cláusula WHERE de una consulta SQL.
strOrderBy	String	Cláusula ORDERBY de una consulta SQL.
strLimitOffset	String	Cláusula LIMITOFFSET de una consulta SQL.
connection	dbConnection	Objeto conexión a la base de datos.
Método	getPageTotal()	
Pre-condición	Los atributos *itemtotal* y *pagenumber* tienen que estar instanciados.	
Descripción	Devuelve el número total de páginas de datos.	
Método	generarSQL()	
Descripción	Genera y devuelve la consulta SQL adecuada según los criterios de búsquedas seleccionados.	
Método	ejecutarSQL()	
Descripción	Ejecuta la consulta SQL que retorna el método *generarSQL* y devuelve el resultado en un arreglo de tuplas.	
Método	__init__()	
Descripción	Instancia un objeto FiltroBD con los atributos vacíos.	

Nombre	Filtro_ReporteIndisciplina	
Descripción	Clase que hereda de FiltroBD para realizar búsquedas de reportes de indisciplinas según determinados criterios.	
Atributos	Tipo	Descripción
fecha_inicial	Date	Atributo para buscar los reportes de las indisciplinas ocurridas desde una fecha.
fecha_final	Date	Atributo para buscar los reportes de las indisciplinas ocurridas hasta una fecha.
facultad	Integer	Atributo para buscar los reportes de las indisciplinas con estudiantes involucrados de una facultad.
existen_trabajadores	Boolean	Toma True para buscar los reportes de las indisciplinas con trabajadores involucrados, False en caso contrario.
sin_sancion	Boolean	Toma True para buscar los reportes de las indisciplinas que no han sido sancionadas, False en caso contrario.

Nombre	ListadorBD	
Descripción	Clase para listar registros de la base de datos.	
Atributos	Tipo	Descripción
connection	dbConnection	Objeto conexión a la base de datos.
Método	getUsuarios()	
Descripción	Retorna en un diccionario todos los nombres de usuarios del sistema con los nombres de las personas correspondientes.	
Método	getTiposSanciones()	
Descripción	Retorna en un diccionario todos los Id. de los tipos de sanciones con las descripciones correspondientes.	

Capitulo 3: Validación de los Datos.

Introducción

Este capítulo trata sobre la integridad relacional de la BD que se propone. En él se realiza un análisis sobre la normalización y redundancia de la información. Se documenta detalladamente para cada tabla las llaves primarias, las llaves foráneas, las columnas que no admiten valores nulos, el chequeo de la información que se inserta y se actualiza en la BD así como los triggers necesarios.

Integridad Relacional.

La implementación de la integridad relacional en una BD consiste en establecer las reglas de consistencia correspondientes a los datos requeridos de las tablas, el chequeo de los valores válidos y únicos, así como la integridad de las llaves primarias y foráneas.

Tipos de Restricciones de Integridad en BD Relacionales.

Datos Requeridos: Establece que una columna tenga un valor no nulo. Se define efectuando la declaración de una columna como NOT NULL cuando la tabla que contiene la columna se crea por primera vez, como parte de la sentencia CREATE TABLE.

Chequeo de Validez: Cuando se crea una tabla cada columna tiene un tipo de datos y el SGBD asegura que solamente los datos del tipo especificado sean ingresados en la tabla.

Integridad de Entidad: Establece que la clave primaria de una tabla debe tener un valor único para cada fila de la tabla, sino la base de datos perderá su integridad. Se especifica en la sentencia CREATE TABLE. El SGBD comprueba automáticamente la unicidad del valor de la clave primaria con cada sentencia INSERT Y UPDATE. Un intento de insertar o actualizar una fila con un valor de la clave primaria ya existente fallara.

Integridad Referencial: Asegura la integridad entre las claves ajenas y primarias (relaciones padre/hijo). Existen cuatro actualizaciones de la base de datos que pueden corromper la integridad referencial:

A continuación se realiza la descripción de dichas reglas de integridad para los diseños de BD que se propusieron.

Módulo Reportes de Indisciplinas.

Tabla	t_reportes				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idr	✓	PK			
c_titulo	✓				
c_fcreado	✓				
c_findisciplina	✓				
c_lugar	✓				
c_existe_trabajador	✓				
c_descripcion	✓				
c_ usuario	✓	FK	t_usuarios	Cascade	Restrict
Columnas Unitarias	c_idr				
Chequeo			Descripción		
(c_fcreado <= CURRENT_DATE) AND (c_fcreado >= c_findisciplina)			La fecha de creado el reporte es menor o igual que la fecha actual y mayor o igual que la fecha en que ocurre la indisciplina.		

Tabla	t_facultades				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_facultad	✓	PK			
Columnas Unitarias	c_facultad				
Chequeo		Descripción			
(c_facultad)::integer > 0		El número de facultad es entero y positivo.			

Tabla	t_ facultades_implicados				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_id	✓	PK			
c_ facultad	✓	FK	t_facultades	Cascade	Restrict
c_idr	✓	FK	t_reportes	Cascade	Cascade
Columnas Unitarias	c_id				
	c_ facultad, c_idr				

Tabla	t_ sancionados				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_id	✓	PKF	t_facultades_implicados	No Action	Cascade
c_ ids	✓	PKF	t_sanciones	Cascade	Restrict
c_cantidad	✓				
Col. Unitarias	c_id, c_ids				
Chequeo		Descripción			
c_cantidad > 0		La cantidad de sancionados es un valor positivo.			

Tabla	t_ sanciones				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_ ids	✓	PK			
c_descripcion	✓				
Columnas Unitarias	c_ids				

Módulo Medios Básicos.

Tabla	t_medios				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idm	✓	PK			
c_descripcion	✓				
c_tipo	✓				
c_precio					
Columnas Unitarias	c_idm				
	c_descripcion				
Chequeo			Descripción		
(c_tipo = 'm') OR (c_tipo = 'e') OR (c_tipo = 'u')			El tipo de medio tiene que ser mueble (m), equipo (e) o útil (u).		

Tabla	t_medios_inventariados				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_id	✓	PK			
c_fecha_alta	✓				
c_idm	✓	FK	t_medios	No Action	Cascade
c_idl		FK	t_locales	No Action	Cascade
Columnas Unitarias	c_id				
Chequeo			Descripción		
c_fecha_alta <= CURRENT_DATE			La fecha de alta de un medio tiene que ser menor o igual que la fecha actual.		

Tabla	t_averias				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_id	✓	PKF	t_medios_inventariados	No Action	Cascade
c_descripción	✓				
Col. Unitarias	c_id				

Tabla	t_utiles					
Columna	No Nulo	Llave	Default	Tabla Foránea	OnUpd.	OnDel.
c_id	✓	PK				
c_bien			0			
c_regular			0			
c_mal			0			
Columnas Unitarias	c_id					
Chequeo			Descripción			
(c_bien > 0) OR (c_regular > 0) OR (c_mal > 0)			Los útiles tienen que tener un estado de bien, regular o mal.			

Tabla	t_aft				
Columna	NoNulo	Llave	Tabla Foránea	OnUpd.	OnDel.
c_id	✓	PKF	t_medios_inventariados	NoAction	Cascade
c_num_inventario					
c_estado	✓				
c_movido_temporal	✓				
Columnas Unitarias	c_id				
	c_num_inventario				

Tabla	t_equipos				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_id	✓	PKF	t_aft	No Action	Cascade
c_num_serie					
c_idmm		FK	t_marcas_modelos	NoAction	Cascade
Columnas Unitarias	c_id				
	c_num_serie				

Tabla	t_marcas_modelos				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idmm	✓	PK			
c_marca					
c_modelo					
Columnas Unitarias	c_idmm				
	c_marca, c_modelo				

Tabla	t_disposiciones				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idm	✓	PKF	t_medios	No Action	Cascade
c_idl	✓	PKF	t_locales	No Action	Cascade
c_disposicion	✓				
Columnas Unitarias	c_idm, c_idl				

Tabla	t_movimientos_temporales				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_id	✓	PKF	t_medios_inventariados	No Action	Cascade
c_idl	✓	PKF	t_locales	No Action	Cascade
c_cantidad	✓				
Col. Unitarias	c_id, c_idl				

Chequeo	Descripción
c_cantidad > 0	La cantidad de medios que se mueven es un valor positivo.

Tabla	t_locales				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idl	✓	PK			
c_observacion					
Columnas Unitarias	c_idl				

Tabla	t_otros_locales				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idl	✓	PKF	t_locales	No Action	Cascade
c_nombre	✓				
c_num_edif		FK	t_edificios	Cascade	Cascade
Columnas Unitarias	c_idl				
	c_nombre, c_num_edif				

Módulo Alojamiento.

Tabla	t_apartamentos				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idl	✓	PKF	t_locales	No Action	Restrict
c_num_apto	✓				
c_habitable	✓				
c_cant_cuartos	✓				
c_capacidad	✓				
c_sexo					
c_facultad		FK	t_facultades	Restrict	Restrict
Columnas Unitarias	c_idl				
	c_num_apto				

Chequeo	Descripción
(c_capacidad > 0) AND (c_cant_cuartos > 0)	La capacidad de un apartamento y la cantidad de cuartos son valores positivos.
c_habitable OR (c_sexo IS NULL)	Un apartamento habitable tiene que tener personas de un sexo sino no es habitable.
(c_sexo IS NULL) OR (c_sexo = 'm') OR (c_sexo = 'f')	Un apartamento puede no tener personas, o tener personas de sexo masculino (m) o femenino (f).

Trigger	Tipo	Descripción
tr_check_edificio	Before insert or update	Chequea en la tabla "t_edificios" si el edificio del apartamento existe.

Tabla	t_edificios				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_num_edif	✓	PK			
c_cant_pasos	✓				
c_cant_pisos	✓				
c_manzana	✓	FK	t_manzanas	Cascade	Cascade
c_facultad	✓	FK	t_facultades	Cascade	Restrict
c_residencia	✓	FK	t_residencias	Cascade	Cascade
Columnas Unitarias	c_num_edif				

Chequeo	Descripción
(c_cant_pasos > 0) AND (c_cant_pisos > 0)	La cantidad de pasos de escaleras y la cantidad de pisos de un edificio son valores enteros y positivos.

Trigger	Tipo	Descripción
tr_eliminar_aptos	Before delete	Elimina en la tabla "t_apartamentos" todos los apartamentos del edificio que se va a eliminar.

Tabla	t_manzanas				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_manzana	✓	PK			
Columnas Unitarias	c_manzana				

Chequeo	Descripción
(c_manzana)::integer > 0	El número de la manzana es un valor entero y positivo.

Tabla	t_residencias				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_residencia	✓	PK			
c_tipo	✓				
Columnas Unitarias	c_residencia				
	c_tipo				

Chequeo	Descripción
(c_residencia)::integer > 0	El número de la residencia es un valor entero y positivo.
(c_tipo = 'e') OR (c_tipo = 'p')	El tipo de residencia es estudiantes (e) o profesores (p).

Tabla	t_residentes				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_id	✓	PK			
c_persona	✓				
c_estado	✓				
c_idl		FK	t_apartamentos	Cascade	Restrict
Columnas Unitarias	c_id				
	c_persona				

Chequeo	Descripción
(c_estado = 'n') OR (c_estado = 'b') OR (c_estado = 'l')	El estado del residente es normal (‘n’), baja (‘b’) o licencia (‘l’).

Trigger	Tipo	Descripción
tr_insertar_accion	After insert or update	Inserta una tupla en “t_historial” con la acción de entrada o salida de una persona en un apartamento.
tr_desubicar	After update	Actualiza a Null la referencia del apartamento si el estado del residente es licencia o baja.

Tabla	t_historial				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_accion	✓				
c_fecha	✓				
c_id	✓	FK	t_residentes	Cascade	Restrict
c_idl	✓	FK	t_apartamentos	Cascade	Restrict
Columnas Unitarias	c_accion, c_fecha, c_id, c_idl				

Chequeo	Descripción
(c_accion = 'e') OR (c_accion = 's')	Una acción es de entrada o de salida en un apartamento.
c_fecha <= CURRENT_DATE	La fecha en que ocurre una acción no puede exceder a la fecha actual.

Módulo Seguridad.

Tabla	t_usuarios				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_usuario	✓	PK			
c_nombre	✓				
c_cargo	✓				
Columnas Unitarias	c_usuario				

Tabla	t_roles				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idr	✓	PK			
c_rol	✓				
c_descripcion	✓				
Columnas Unitarias	c_idr				
	c_rol				

Tabla	t_roles_usuarios				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_usuario	✓	PKF	t_usuarios	Cascade	Cascade
c_idr	✓	PKF	t_roles	No Action	Restrict
c_residencia					
c_facultad					
Columnas Unitarias	c_usuario, c_idr				

Chequeo	Descripción
(c_residencia IS NOT NULL) OR (c_facultad IS NOT NULL)	El número de residencia y la facultad no pueden tener valores nulos al mismo tiempo.

Tabla	t_permisos_roles				
Columna	No Nulo	Llave	Tabla Foránea	OnUpdate	OnDelete
c_idr	✓	PKF	t_roles	No Action	Restrict
c_permiso	✓				
c_modulo	✓				
Columnas Unitarias	c_idr				
Chequeo	Descripción				
(c_permiso = 0) OR (c_permiso = 1)	El permiso es de lectura (0) o lectura/escritura (1).				
(c_modulo = '1') OR (c_modulo = '2')	El módulo es Alojamiento (1) o Medios Básicos (2).				

Normalización y Análisis de Redundancia.

Los esquemas de base de datos propuestos están normalizados hasta la Tercera Forma Normal con el objetivo de eliminar las redundancias y las anomalías de inserción, eliminación y actualización, aunque tampoco se viola la forma normal de Boyce-Codd. Se puede comprobar que se cumple con la Primera Forma Normal porque las entradas de los registros almacenan valores escalares. También se cumple con la Segunda Forma Normal porque los diseños además de estar en Primera Forma Normal, todos los atributos no primos de las relaciones dependen totalmente de la llave primaria correspondiente. Y por último se cumple con la Tercera Forma Normal porque se cumple con la Segunda Forma Normal y no existen dependencias transitivas de atributos no primos con llaves candidatas.

En el modelo de datos están presentes tres tablas que no poseen un identificador único debido a sus características. La tabla t_historial en el módulo Alojamiento registra información de las entradas y las salidas de personas en los apartamentos de la Residencia, y ya que dichos registros nunca serán actualizados no es necesario definir una llave primaria. En las tablas t_roles_usuarios y t_permisos_roles del módulo Seguridad se almacenan pocos datos por lo que no se obtiene una eficiencia adicional con la utilización del índice asociado a una llave primaria en el momento de realizar selecciones y actualizaciones.

Conclusiones

Una vez culminadas las tareas de investigación y desarrollo que se planificaron se llegó a la conclusión de que el objetivo fundamental del trabajo fue logrado. El diseño de la Base de Datos para los módulos priorizados del proyecto: Reportes de Indisciplinas, Medios Básicos y Alojamiento, así como el acceso a los datos desde Zope permite viabilizar la gestión automática de la información en la aplicación web de la Residencia de la UCI.

Se considera además que el presente trabajo significa un aporte al quehacer investigativo y productivo de la UCI porque:

- Se estimuló el uso de PostgreSQL como SGBD en proyectos de pequeño y mediano formato.
- Se estimuló el uso de Zope como framework de desarrollo web y Python como lenguaje de programación.
- Se logró la integración de Zope con PostgreSQL a través de un producto implementado en Python.
- Se comprobó el uso de psycopg2 como driver de comunicación entre Python y PostgreSQL.
- Se aplicaron buenas prácticas de diseño mediante el uso de patrones de acceso a datos para aplicaciones orientadas a objetos.
- Se obtuvieron los diseños de la Base de Datos en Tercera Forma Normal.
- Se garantizó la integridad de la información en la Base de Datos.

Recomendaciones

Todo lo expuesto en este documento pretende ser la base sobre la cual continuar la implementación de la Base de Datos y el acceso a datos para los demás módulos colocando en primer lugar los módulos de Alojamiento y Medios Básicos puesto que constituyen los módulos de mayor prioridad.

Se debe asumir el diseño de BD de los restantes módulos que por su nivel de prioridad no fueron atendidos en las iteraciones que hasta el momento se han efectuado. Se deben implementar los procedimientos almacenados y triggers que provean los datos necesarios para la aplicación y su consistencia. El diseño del acceso a datos para los módulos que faltan debe corresponderse con el diseño del módulo Reportes de Indisciplinas, pues de esta manera solo se necesitará diseñar los objetos activos del negocio.

Se debe diseñar también las clases necesarias para interactuar con los servicios web de las bases de datos externas que ofrecen información complementaria a la aplicación. Por lo que se propone diseñar una clase que gestione las conexiones hacia los servicios web, además de las clases proxy que permitan su consumo. Al mismo tiempo se señala que se deben implementar servicios web que brinden información de la base de datos local que es de interés para las demás aplicaciones desarrolladas en la UCI.

Para incorporar la BD que se propone en la producción o la práctica social se recomienda integrarla en una aplicación web para que toda la comunidad universitaria de la UCI pueda acceder a su información. Este trabajo tiene la propiedad de ser extensible hacia otros dominios de aplicación. Se puede afirmar que los diseños realizados modelan procesos de negocio que cualquier entidad que controle medios y gestione servicios de alojamiento puede tener.

Por último se recomienda establecer una política de seguridad ante fallos y recobrado de la información en la BD. Se pueden generar de manera automática backups periódicos así como la realización de tareas de mantenimiento.

Bibliografía

1. Base de Datos. 2007. [2007]. Disponible en: http://es.wikipedia.org/wiki/Base_de_datos
2. GARCIA, R. M. M. Diseño de Bases de Datos, 1999.
3. Los sistemas de bases de datos y los SGBD. 2004. [2007]. Disponible en: http://tramullas.com/documatica/
4. SQL. 2007. [2007]. Disponible en: http://es.wikipedia.org/wiki/SQL
5. Accessing Zope from PostgreSQL. Disponible en: http://www.zope.org/Members/pupq/zope_in_pg
6. Comunidad en Chile de Plone, Zope y Python. Disponible en: http://www.plonechile.cl/
7. Curso Zope. Disponible en: http://www.plope.com/
8. Diseñadores para PostgreSQL. Disponible en: http://www.postgresql.org/download/commercial
9. Documentación de Python. Disponible en: http://sourceforge.net/project/showfiles.php?group_id=9845
10. Documentación de Python-es. Disponible en: http://pyspanishdoc.sourceforge.net/
11. Foro de las comunidades en la UCI. Disponible en: http://foro.prod.uci.cu
12. FTP wingware - wing ide. Disponible en: ftp://wingware.com/pub/wingide/
13. Generadores de código. Disponible en: http://www.codegeneration.net/
14. Información de Python - es. Disponible en: http://listas.aditel.org/listinfo/python-es
15. Libro de PostgreSQL. Disponible en: http://book.itzero.com/read/others/0508/Sams.PostgreSQL.2nd.Edition.Jul.2005_htm
16. Página principal de MSDN en Español. Disponible en: http://www.microsoft.com/spanish/msdn/spain/default.mspx
17. pgAdmin III PostgreSQL. Disponible en: http://www.pgadmin.org/
18. PostgreSQL Replicator. Disponible en: http://pgreplicator.sourceforge.net/
19. Projects psycopg1. Disponible en: http://initd.org/projects/psycopg1
20. Python 2.5 - Documentación. Disponible en: http://docs.python.org/
21. Python Cheese Shop Browse. Disponible en: http://cheeseshop.python.org/pypi?:action=browse&c=258&c=256
22. Python Database Modules. Disponible en: http://www.python.org/topics/database/modules.html
23. Python Web Services. Disponible en: http://pywebsvcs.sourceforge.net/
24. Sitio de BDs. Disponible en: http://www.downloaddatabase.com/
25. Sitio de herramientas Case. Disponible en: http://www.casestudio.com/

26. Sitio de software libre en la UCI. Disponible en: http://softwarelibre.uci.cu/modules/news/
27. Sitio de soluciones DTM de BDs. Disponible en: http://www.sqledit.com/
28. Sitio de Zope. Disponible en: http://www.zope.org/
29. The slony1 Project. Disponible en: http://slony1.projects.postgresql.org/
30. Command Site. Disponible en: http://www.commandprompt.com/
31. zopewiki.org Products. Disponible en: http://zopewiki.org/Products

Indice de Figuras

Páginas

Glosario de Términos

ACID

Denominación que recibe todo Sistema Gestor de Bases de Datos que realiza transacciones seguras.

ANSI SQL

Estándar de codificación SQL para sistemas gestores de bases de datos.

CDIR

Es un patrón de resumen de rutas consistente en la simplificación de varias direcciones de redes o subredes en una sola dirección IP.

GiST

Tipo de índice en PostgreSQL para el trabajo con datos geométricos.

IPv4 - IPv6

Protocolos de comunicación que identifican con un número único a las computadoras conectadas a la Internet o a una Intranet corporativa.

JDBC (Java Database Connectivity)

Marco de programación para los desarrolladores de aplicaciones Java que tienen acceso a la información guardada en bases de datos, hojas de calculo y archivos "planos".

Kerberos

Protocolo de autenticación de redes de ordenador que permite a dos computadores en una red insegura demostrar su identidad mutuamente de manera segura.

MVCC (Multiversion Concurrency Control)

Técnica empleada en PostgreSQL para gestionar el acceso concurrente a los datos y garantizar la consistencia de las bases de datos.

ODBC (Open Database Connectivity)

Estándar de acceso a bases de datos desarrollado por Microsoft Corporation cuyo objetivo es hacer posible el acceso a cualquier dato desde cualquier aplicación, sin importar qué SGBD almacene los datos.

SHA1

Algoritmo de reducción criptográfico de 160 bits de un mensaje que puede tener un tamaño máximo de 2^{64} bits, y se basa en principios similares a los usados en el diseño de los algoritmos de resumen de mensajes MD4 y MD5.

WebDAV

Estándar que describe cómo a través de la extensión del protocolo HTTP 1.1 pueden realizarse acciones de gestión de archivos tales como escribir, copiar, eliminar o modificar.

XMLRPC

Protocolo de llamada a procedimiento remoto que usa XML para codificar las llamadas y HTTP como mecanismo de transporte.