Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Licenciatura en Ciencias de la Computación

Tratamiento de Restricciones de Integridad para OORDB.

Caso de estudio: PostgreSQL.

Tesis

Que para obtener el Título de
Licenciado en Ciencias de la Computación

Presenta

José Javier Juárez Caballero

Asesor

Lic. Ma. del Rocío Boone Rojas

Fac. de Ciencias de la Computación, BUAP.

Coasesor

Ing. Oscar Eduardo Pérez Carrasco

Vicerrectoría de Docencia, BUAP.

Puebla, Pue.
2008

DEDICATORIA

Al creador, por permitirme cumplir este sueño.
A mi familia y amigos por animarme en los momentos difíciles. Lore, Andy y bebe los amó.

Agradecimientos

*Me gustaría dar las gracias a mi amigo el **Ing. Oscar Eduardo Pérez Carrasco**, por haber confiado en mí y haberme dado las herramientas necesarias para adentrarme en este universo virtual. También quiero agradecer a mi asesora de tesis la **Mtra. María del Rocío Boone Rojas** por tener la paciencia y apoyarme para terminar este proyecto.*

Finalmente quiero agradecer a mis padres que siempre han creído en mí.

ÍNDICE

INTRODUCCIÓN **1**

CAPÍTULO I
Integridad en base de datos **3**

1.1 El concepto de Integridad……. 3
1.2 Representación de restricciones de Integridad…….. 4
1.3 Tratamiento de Integridad……. 5
1.4 Restricciones de Integridad en bases de datos relacionales……..... 5
1.4.1 Información faltante…….. 6
1.4.2 Restricciones de integridad elementales……. 6
1.4.3 Triggers……... 7
1.4.4 Restricciones de integridad particulares……. 8

CAPÍTULO 2
Bases de datos orientadas a objetos (OODB) **9**

2.1 Antecedentes…….. 9
2.1.1 Lenguajes de programación persistentes…… 10
2.2 Objetos……. 11
2.2.1 Estructura de un objeto……. 11
2.2.2 Encapsulamiento…… 12
2.2.3 Organización de los objetos……. 12
2.2.4 Polimorfismo……. 13
2.2.5 Identidad de los objetos……... 13
2.3 Clases……. 13
2.4 Herencia…… 15
2.4.1 Herencia múltiple…….. 16
2.5 Métodos……... 18

CAPÍTULO 3
Bases de datos relacionales orientadas a objetos (OORDB) **19**

3.1 Modelos OORDB…….…… 19
3.1.1 Tipos complejos……. 19
3.1.2 Tipos estructurados……. 20
3.1.3 Creación de valores de tipos complejos…… 21
3.1.4 Herencia…….. 22
3.1.4.1 Herencia de tipos……. 22
3.1.4.2 Herencia de tablas……. 23
3.2 Comparaciones entre OODB y OORDB…….. 23
3.3 Clasificación de sistemas manejadores de bases de datos……. 24
3.3.1 Datos simples sin consulta a la DB……. 25
3.3.2 Datos simples y consulta a la DB……. 25

3.3.3 Datos complejos sin consulta a la DB……..... 27
3.3.4 Datos complejos y consulta a la DB……:……. 30

CAPÍTULO 4
POSTGRESQL **34**

4.1 PostgreSQL…….. 34
4.2 Otras características de PostgreSQL…… 36
4.3 Implantación de una OORDB experimental…….. 38
4.3.1 Descripción del problema……. 38
4.3.2 Diagrama E-R..... 39
4.3.3 Grafo acíclico dirigido…… 40
4.3.4 Diccionario de datos……. 41
4.3.5 Modelo relacional……. 44
4.3.6 Normalizado de tablas…… 46

CAPÍTULO 5
Tratamiento de restricciones de integridad en POSTGRESQL **47**

5.1 Resultados de la evaluación de integridad……. 47
5.2 Análisis de los resultados de la evaluación de integridad…….. 49

CONCLUSIONES……... **69**
BIBLIOGRAFÍA…… **71**
GLOSARIO……... **73**

ÍNDICE DE FIGURAS Y TABLAS

FIGURA 2.4.1.	Jerarquía de clases……..	15
FIGURA 2.4.1.1	GAD de clases…….	17
TABLA 3.1	Una clasificación de Aplicaciones DBMS…….	24
FIGURA 3.2	Arquitectura Estándar Cliente – Servidor……..	27
FIGURA 3.3	La Arquitectura de los lenguajes Persistentes……..	29
FIGURA 4.1	Diagrama E-R……..	38
TABLA 4.2	Grafo Acíclico Dirigido…….	39
TABLA 4.3	Tabla de relaciones……..	41
TABLA 4.4	Tabla de entidades…….	42
TABLA 4.5	Modelo relacional…….	44
TABLA 4.6	Normalizado de tablas…….	45
TABLA 5.1	Pruebas de Integridad de Entidades……..	47
TABLA 5.2	Pruebas de Integridad Referencial…….	48
TABLA 5.3	Pruebas de Restricciones de Rechazo……..	48
TABLA 5.4	Pruebas de otras restricciones…….	48
FIGURA 5.5	Tabla Vehículo……..	49
TABLA 5.6	Llaves primarias y foráneas…….	49
FIGURA 5.7	Sugerencia de creación de llave foránea…….	50
FIGURA 5.8	Intento de insertar un valor nulo en un campo de llave foránea……..	51
FIGURA 5.9	Intento de insertar una cadena vacía en un campo de llave primaria…….....	51
FIGURA 5.10	Cadena vacía en un campo de llave primaria…….	51
FIGURA 5.11	Intento de insertar un valor repetido en un campo de llave primaria…….....	51
FIGURA 5.12	Intento de insertar un valor nulo en un campo de llave primaria de más de un componente……..	52
FIGURA 5.13	Intento de insertar valores repetidos en un campo de llave primaria…….	52
FIGURA 5.14	Actualización en un campo de llave primaria por un valor nulo……..	53
FIGURA 5.15	Intento de actualizar un campo de llave primaria por un valor repetido……..	53
FIGURA 5.16	Intento de actualizar un campo de llave primaria de más de un componente por un valor nulo……..	54
FIGURA 5.17	Intento de actualizar un campo de llave primaria de más de un componente por una combinación de valores repetidos……..	54
FIGURA 5.18	Verificación de Integridad Referencial……..	55
FIGURA 5.19	Eliminación de tuplas restringida…….	56
FIGURA 5.20	Eliminación de tuplas en cascada…….	57
FIGURA 5.21	Eliminación de tuplas con valores puestos a nulos…….	58

FIGURA 5.22	Actualización restringida……	59
FIGURA 5.23	Actualización en cascada……	60
FIGURA 5.24	Actualización con valores puestos a nulos……...	61
TABLA 5.25	Tabla Arrendatario con restricción Check……	62
FIGURA 5.26	Restricción Check……..	62
TABLA 5.27	Valores únicos……..	63
TABLA 5.28	Valores por default……..	63
TABLA 5.29	Tipos de datos definidos por el usuario……..	64
TABLA 5.30	Función Trigger……	65
TABLA 5.31	Disparador……	65
FIGURA 5.32	Prueba con disparadores……	66
FIGURA 5.33	Reglas de PostgreSQL……	67
FIGURA 5.34	Ejecución de la Regla de PostgreSQL……	68

INTRODUCCIÓN

El hombre se ha visto en la necesidad de procesar gran cantidad de información. Primero lo hizo manualmente, almacenándola en libros, cuadernos o en hojas, esto hacia que su manipulación fuera muy costosa en tiempo y complejidad, debido a que la búsqueda y consulta de la información se hacía teniendo que localizar la evidencia física.

Las primeras bases de datos surgieron del desarrollo de los sistemas de gestión de archivos. Estos sistemas primero evolucionaron en bases de datos de red ó en bases de datos jerárquicas y, más tarde, en bases de datos relacionales.

A partir de la época de los 70's, al introducir Edgar F. Codd el Modelo Relacional, ha sido de gran utilidad y tenido una gran aceptación. Los datos se representan por medio de tablas con el objetivo de evitar inconsistencia y redundancia en la información.

Las Bases de Datos Orientadas a Objetos (OODB) surgieron en un principio para soportar la programación Orientada a Objetos (OO), y almacenar lo que se conoce como datos persistentes. El modelo relacional no se basa en un paradigma para la estructuración de los datos, sino en ciertos fundamentos matemáticos.

Las restricciones de integridad verifican que los datos que se ingresan o se actualizan en la base de datos sean consistentes, además, de protegerla contra accesos sin autorización y daños accidentales.

En este trabajo se considera como antecedente al modelo relacional como fundamento central y sus especificaciones complementarias para la regla de integridad referencial de tal modelo.

Como ya se ha señalado, la integridad de los datos es fundamental, sin embargo, los sistemas de objetos, no soportan generalmente restricciones de integridad declarativas; en vez de ello, requieren que tales restricciones se hagan cumplir por medio de código procedural (es decir, por métodos ó posiblemente por programas de aplicación).

Se desea realizar un análisis del tratamiento al problema de integridad de bases de datos orientadas a objetos tomando como caso de estudio a POSTGRESQL.

Frecuentemente las restricciones de integridad se aplican en la aplicación en lugar de integrarlas en el diseño de la base de datos, se pretende mostrar y evaluar el funcionamiento de estas restricciones basándonos en un modelo de evaluación que se citará en el capítulo 5.

Objetivo General.

Identificar e implantar los mecanismos que ofrece POSTGRESQL para el tratamiento de restricciones de integridad para OORDB.

Objetivos Específicos.

1. Documentar y experimentar las facilidades que ofrece POSTGRESQL para el manejo de OORDB.
2. Desarrollar e implantar los métodos necesarios para ofrecer un soporte para el componente de integridad de POSTGRESQL para OORDB.

Organización de la tesis

- Capítulo 1
  - En este capítulo, se muestra una panorámica general acerca del concepto de Integridad y de las reglas de integridad en bases de datos relacionales. Se documentan los tipos de restricciones elementales y particulares.
- Capítulo 2
  - En este capítulo, se realiza una descripción del modelo de bases de datos orientado a objetos. Detallando el concepto de objeto, propiedades, métodos, herencia, polimorfismo, encapsulamiento y clases.
- Capítulo 3
  - En este capítulo se describe el modelo relacional orientado a objetos, agregando una comparación entre el modelo de archivos, relacional, de objetos y relacional orientado a objetos, así como, un ejemplo del tipo de ámbito recomendado para cada uno de los esquemas.
- Capítulo 4
  - En este capítulo, se detallan las características del manejador de base de datos. Más adelante se agrega la documentación de una base de datos experimental sobre una agencia de alquiler de autos, en la cual se evaluará el desempeño de PostgreSQL.
- Capítulo 5
  - En este capítulo, utilizando el marco de evaluación del trabajo de tesis propuesto, se realizó una evaluación del funcionamiento de las restricciones de integridad del manejador en la base de datos experimental.

CAPÍTULO 1
Integridad en base de datos

La integridad es una cualidad que garantiza la calidad de los datos.

1.1 El concepto de Integridad

El concepto de integridad se refiere al aseguramiento de que la información contenida en la base de datos sea correcta.

La integridad implementa las medidas de protección que se incluyen en un sistema de información para evitar la pérdida accidental de los datos cuyo propósito es informar al DBMS de ciertas restricciones del mundo real (por ejemplo, que la edad de un empleado debe ser mayor ó igual que 18 años) para evitar la inconsistencia de la información.

Una restricción de Integridad es una propiedad que la Base de Datos debe satisfacer en cualquier instante.

La mayoría de las bases de datos deben estar sujetas a ciertas reglas de integridad.

Hay dos tipos de restricciones:

- ***Estáticas:*** Comprenden desde simples restricciones de dominio (por ejemplo, la edad de una persona debe ser un valor entre 0 y 120 años) o hasta complejas relaciones entre diferentes piezas de información (por ejemplo, los proyectos dirigidos por un investigador deben ser aquellos en los que el investigador tiene asistentes de investigación).
- ***Dinámicas:*** Son aquellas que restringen las posibles transiciones de estado de la Base de Datos (por ejemplo, los salarios no pueden bajar).

Existen dos formas de incorporar el tratamiento de restricciones de integridad en el sistema:

- Las aplicaciones que actualizan la base de datos pueden incorporar código adicional que verifique y asegure que no se violen las restricciones.
- El diseñador de la base de datos puede declarar restricciones como parte del esquema. El DBMS debe verificar automáticamente las restricciones cada vez que se actualicen los datos.

Como principio general, la especificación de restricciones de integridad debería ser excluida de las aplicaciones para reducir los costos (tiempo, código adicional) de la programación y para asegurar la calidad en los datos.

Incluir las restricciones de integridad en la base de datos, es decir, que el DBA las especifique y el DBMS las asegure, trae consigo las siguientes ventajas:

- Reduce los costos de desarrollo de software debido a que las restricciones de integridad son codificadas solo una vez y compartidas por todos los usuarios.
- El control de las restricciones de integridad es más confiable por ser centralizado y uniforme.
- Hace más fácil el mantenimiento. El DBA es el único encargado de definir y modificar las restricciones de integridad.

1.2 Representación de las restricciones de integridad.

Las principales restricciones estáticas pueden clasificarse en:

- ***Restricciones de cardinalidad:*** restringen la cantidad de objetos que puede referenciar a través de un atributo o a la cantidad de objetos agregados que lo pueden referenciar. La especificación de la mínima cantidad de objetos referenciados trae consigo cuestiones semánticas adicionales; si es 0 es un atributo opcional y si es 1 es un atributo obligatorio.
  Cuando un objeto tiene un atributo opcional surge la necesidad de especificar si se permite o no asociar una referencia nula, la cual en términos de implantación normalmente se trata permitiendo o no que el atributo tenga un indicador permitido por el sistema (indicador NULL) y que represente información faltante.
- ***Restricciones de dominio:*** solo los objetos especificados pueden servir como dominio de un atributo.
- ***Restricciones de unicidad:*** aseguran que un objeto puede ser identificable usando un determinado atributo. Al respecto, es importante que exista la distinción entre el objeto y el(los) nombre(s) usado(s) para identificarlo.
- ***Restricciones de coexistencia:*** una instancia de la clase hija debe también existir como instancia de sus clases padre.
- ***Restricciones de clases hijas disjuntas:*** las instancias de la clase padre solo pueden pertenecer a una clase hija dentro de la jerarquía.
- ***Restricción de cobertura:*** todas las instancias de la clase padre deben ser instancias de al menos una clase hija. Por defecto se permite que la unión de las clases hijas no sea igual a la clase padre, es decir, que existan instancias de la clase padre que no son instancias de alguna clase hija.

1.2 Tratamiento de integridad

La forma más sencilla de comprobar restricciones estática/dinámicas es evaluar cada una de ellas después de la transacción, sin embargo, esto puede ser muy costoso en bases de datos voluminosas, ya que no se aprovecha el hecho de que la base de datos era integra antes de la transacción.

Basándose en lo anterior, en bases de datos relacionales y bases de datos deductivas, existen métodos que simplifican la comprobación de la integridad, evitando comprobar instancias de las restricciones que se satisfacían antes de la transacción y que no son afectadas por está.

Otro enfoque para controlar la integridad es ejecutar un programa de verificación que es ejecutado por un determinado evento, tal como inserción o borrado (trigger o disparador). Tanto el programa como la especificación de los eventos relevantes es responsabilidad del usuario (preferentemente el DBA).

Los disparadores o triggers permiten adecuar el comportamiento del gestor de la base de datos a necesidades concretas, por ejemplo, en cualquiera de los siguientes sentidos.

- Generando valores derivados para algunos atributos a partir de los valores de otros.
- Impidiendo transacciones inválidas, según ciertas reglas establecidas.
- Modificando el tratamiento de la integridad realizado por defecto.
- Asegurando autorizaciones de seguridad complejas.
- Manteniendo un replicado de datos transparente.
- Reuniendo estadísticas de acceso a los datos. [13]

1.3 Restricciones de integridad en bases de datos relacionales

Los DBMS actuales están lejos de proporcionar un total y satisfactorio soporte para representación y comprobación de restricciones de integridad. En el modelo relacional y por consiguiente en los DBMS basados en él, se tienen tres restricciones de integridad, aplicables a cualquier base de datos, adicionales a aquellas específicas para cada base de datos. Estas restricciones de integridad están siendo soportadas por algunos DBMS actualmente en el mercado. Estas tres reglas se refieren, respectivamente, a las claves primarias, claves foráneas y los dominios de valores de los atributos.

Antes de describir las mencionadas restricciones de integridad, se introducirá brevemente la problemática relacionada con la información faltante, representada comúnmente en sistemas relacionales con el indicador NULL.

1.3.1 Información faltante

En el mundo real surge con frecuencia el problema de la información faltante. Por ejemplo, los registros históricos contienen a veces entradas tales como “fecha de nacimiento desconocida”.

En la mayoría de los sistemas SQL actuales, la información faltante puede representarse por un indicador especial llamado NULL. Si una tupla tiene un nulo en un atributo, significará que se desconoce el valor de ese atributo ó que quizá no es aplicable en dicha tupla. Es importante señalar que no es lo mismo un nulo que (por ejemplo) un espacio en blanco o un cero; de hecho, no se trata en realidad de un valor de un dato en el sentido usual, sino un identificador de información faltante.

La posibilidad para definir atributos que permitan nulos provocan bastantes problemas que deben tenerse presentes al elaborar las consultas a la Base de Datos. [13]

1.3.2 Restricciones de integridad elementales

a) Integridad de entidades: Ningún componente de la clave primaria de una relación puede aceptar valores nulos.

Si en una tupla algún atributo que forma parte de la clave primaria es nulo significaría que la entidad en cuestión no es identificable y no tiene sentido guardar información de algo que no se puede identificar.

b) Integridad referencial: la base de datos no debe contener valores de clave foránea sin concordancia, es decir, que el valor al que se hace referencia no exista o no este relacionado con el anterior. Para cada clave foránea es necesario responder tres preguntas:

- ¿Pueden aceptar valores nulos?
- ¿Qué debe suceder si hay un intento de eliminar la tupla con la clave primaria a la cual referencia alguna clave foránea? Se presentan las siguientes alternativas:
  - Restringir: rechazar la eliminación si existe alguna tupla con esa clave foránea.
  - Propagar: eliminar la tupla con clave primaria y todas las tuplas con esa clave foránea.
  - Poner valores nulos: poner un valor nulo en el atributo de todas las tuplas con esa clave foránea, siempre y cuando no exista alguna restricción en el campo de llave foránea que impida poner valores nulos y eliminar la tupla con clave primaria.

- ¿Qué debe suceder si hay un intento de modificar la clave primaria a la cual referencia la clave foránea? Se presentan las siguientes alternativas:

  - Restringir: rechazar la modificación si existe alguna tupla con esa clave foránea.
  - Propagar: modificar la clave primaria y todas las claves foráneas correspondientes.
  - Poner valores nulos: poner un valor nulo en la clave foránea de todas las tuplas correspondientes, siempre y cuando no exista alguna restricción en el campo de llave foránea que impida poner valores nulos y modificar la clave primaria.

c) Integridad de dominio: los valores de los atributos deben estar en el dominio especificado. A cada atributo se le asocia un tipo de dato como dominio de sus valores.

Además para cada atributo puede indicarse que no permite valores nulos poniendo NOT NULL, de lo contrario se asume que puede tener un valor nulo.

Cada DBMS tiene su sintaxis particular para especificar las restricciones de integridad, es necesario consultar el esquema de la Base de Datos. [13]

1.4.3 Disparadores

Son bloques almacenados asociados a una tabla que se ejecutan o disparan automáticamente cuando se producen ciertos eventos sobre la tabla (inserción, borrado o modificación de tupla).

Se utilizan para:
– Implementar restricciones complejas de seguridad o integridad.
– Prevenir transacciones erróneas.
– Gestionar réplicas remotas de la tabla.

Se puede especificar que el disparador se ejecute de cualquiera de estas dos formas: ANTES (BEFORE) de que la operación sea intentada en un registro (antes de que las restricciones se comprueben e INSERT, UPDATE y DELETE sean intentados) o DESPUÉS (AFTER) de que la operación haya sido intentada (por ejemplo, después de que las restricciones sean comprobadas y de que INSERT, UPDATE y DELETE hayan sido completados). Si el disparador se pone en marcha antes del evento, este puede saltar la operación para el registro actual o cambiar el registro que estaba insertándose (solo para las operaciones INSERT y UPDATE) o si se dispara después del evento, todos los cambios, incluyendo la última inserción, actualización o borrado, son visibles para el disparador. [19]

1.4.4 Restricciones de integridad particulares

La restricción NOT NULL especifica una regla que obliga que un campo contenga únicamente valores no nulos. Ésta es únicamente una restricción de campo, y no se permite como restricción de tabla.

- Restricción UNIQUE

La restricción UNIQUE especifica una regla que obliga a uno o más campos de una tabla que deben contener valores únicos.

Las definiciones de campo de las columnas especificadas no tienen porque incluir una restricción NOT NULL para ser incluidos en una restricción UNIQUE.

Cada restricción de campo UNIQUE debe nombrar un campo que es distinto del conjunto de campos nombrados por cualquier otra restricción UNIQUE o PRIMARY KEY definidos por la tabla.

- Restricción CHECK

La restricción CHECK especifica una restricción sobre los valores permitidos en un campo. La restricción CHECK se permite también como restricción de tabla.

CAPÍTULO 2
Bases de datos orientadas a objetos (OODB)

Algunas autores creen que los sistemas de bases de datos orientados a objetos dominarán el mundo y reemplazarán por completo a los Sistemas de bases de datos relacionales, pero otros creen que solo son adecuados para determinados problemas muy específicos y nunca capturarán mas que una pequeña fracción del mercado total. Más recientemente han comenzado a aparecer sistemas que soportan una tercera vía: Sistemas que integran las tecnologías de objetos y las relacionales en un intento por obtener lo mejor de ambos mundos. [3]

2.1 Antecedentes

El deseo de representar objetos complejos ha sido el resultado de la realización de los sistemas orientados a objetos. SMALLTALK, es un lenguaje de programación que fue diseñado específicamente para ser orientado a objetos. Otros lenguajes de programación que son orientados a objetos son C++ y Java.

Los Sistemas de bases de datos orientados a objetos tienen sus orígenes en los lenguajes de programación orientados a objetos. El objetivo es tener la posibilidad de manejar objetos y operaciones sobre esos objetos, que se asemeja mucho más a sus contrapartes en la realidad. La idea fundamental es elevar el nivel de abstracción.

Elevar el nivel de abstracción es incuestionablemente valioso y el paradigma de objetos ha tenido mucho éxito para satisfacer ese objetivo en los lenguajes de programación. Por lo tanto, tiene sentido preguntarnos si el mismo paradigma puede ser aplicado satisfactoriamente en las bases de datos.

Además la idea de manejar una base de datos que está compuesta por objetos es más atractiva desde el punto de vista del usuario, al menos a simple vista.

Los lenguajes de programación y la administración de bases de datos tienen mucho en común, también difieren en determinados aspectos importantes:

- Un programa de aplicación esta hecho para resolver algún problema específico.
- Una base de datos esta hecha, para resolver una variedad de problemas diferentes, y algunos de ellos ni siquiera son conocidos al momento de diseñarla.

Mucha gente cree que las técnicas de bases de datos orientadas a objetos son el enfoque a escoger por las siguientes áreas de aplicaciones:

- CAD/CAM (Diseño y manufactura asistidos por computadora).
- CIM (Manufactura integrada por computadora).
- CASE (Ingeniería de software asistida por computadora).
- GIS (Sistemas de información geográfica).
- Ciencia y medicina.
- Almacenamiento y recuperación de documentos.

A pesar del hecho de que las bases de datos orientadas a objetos fueron pensadas originalmente para aplicaciones complejas, si quieren ser dignas de reconocimiento, también deberán ser capaces de manejar aplicaciones simples. [2]

2.1.1 Lenguajes de Programación Persistentes

Los lenguajes de bases de datos se diferencian de los lenguajes de programación tradicionales en que trabajan directamente con datos persistentes, es decir, los datos que siguen existiendo una vez que el programa que los creo ha concluido.

Mientras que los lenguajes para el tratamiento de datos como SQL son bastante efectivos en el acceso a los datos, se necesita un lenguaje de programación para implementar otros componentes de las aplicaciones como las interfaces de usuario o la comunicación con otras computadoras. La manera tradicional de realizar las interfaces de las bases de datos con los lenguajes de programación es incorporar SQL dentro del lenguaje de programación.

Los lenguajes de programación persistente son lenguajes de programación extendidos con constructores para el tratamiento de datos persistentes. Los lenguajes de programación persistente pueden distinguirse de los lenguajes de programación con SQL incorporado de al menos dos maneras:

- En los lenguajes incorporados el sistema de tipos del lenguaje anfitrión suele ser diferente del sistema de tipos del lenguaje para el tratamiento de los datos. Por el contrario, en los lenguajes de programación persistentes, el lenguaje de consulta se halla totalmente integrado con el lenguaje anfitrión y ambos comparten el mismo sistema de tipos.
- Los programadores que utilizan lenguajes de consulta incorporados son responsables de la escritura de código explicito para la búsqueda de los datos en la memoria. Si se realizan actualizaciones, los programadores deben escribir código de manera explicita para volver a guardar los datos actualizados en la base de datos. Por el contrario, en los lenguajes de programación persistentes los programadores pueden trabajar con datos persistentes sin tener que escribir código para buscarlos en la memoria o para volver a guardarlos en el disco. [13]

2.2 Objetos

El elemento fundamental en una OODB es, como su nombre lo indica, el objeto. Podemos definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

2.2.1 Estructura de un objeto

El paradigma orientado a objetos esta basado en el encapsulamiento de los datos y el código relacionados con cada objeto en una sola unidad cuyo contenido no es visible desde el exterior. [13]

Un objeto puede considerarse como una especie de cápsula dividida en tres partes:

- Relaciones.
- Propiedades.
- Métodos.

Cada uno de estos componentes desempeña un papel totalmente independiente:

Las relaciones permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos.

Las propiedades distinguen a un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

Las interacciones entre cada objeto y el resto del sistema se realizan mediante mensajes. El término mensaje en un entorno orientado a objetos no implica el uso de mensajes físicos en redes informáticas. Por el contrario, hace referencia al intercambio de solicitudes entre los objetos independientemente de los detalles concretos de su implementación. Se utiliza a veces la expresión *invocar a un método* para denotar el hecho de enviar un mensaje a un objeto y la ejecución del método correspondiente. [13]

Los métodos son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

2.2.2 Encapsulamiento

Cada objeto es una estructura compleja en cuyo interior hay datos y programas, todos ellos relacionados entre sí, como si estuvieran encerrados conjuntamente en una cápsula. Esta propiedad es una de las características fundamentales en la OOP.

Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible. Esta propiedad de los objetos se denomina ocultación de la información.

Esto no quiere decir, sin embargo, que sea imposible conocer lo necesario respecto a un objeto y a lo que contiene. Si así fuera no se podría hacer gran cosa con él. Lo que sucede es que las peticiones de información a un objeto. Deben realizarse a través de mensajes dirigidos a él, con la orden de realizar la operación pertinente. La respuesta a estas órdenes será la información requerida, siempre que el objeto considere que quien envía el mensaje está autorizado para obtenerla.

El hecho de que cada objeto sea una cápsula facilita enormemente que un objeto determinado pueda ser transportado a otro punto de la organización, o incluso a otra organización totalmente diferente que precise de él. Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la OOP sea muy apta para la reutilización de programas.

2.2.3 Organización de los objetos

En principio, los objetos forman siempre una organización jerárquica, en el sentido de que ciertos objetos son superiores a otros de cierto modo.

Existen varios tipos de jerarquías: serán simples cuando su estructura pueda ser representada por medio de un "árbol". En otros casos puede ser más compleja.

En cualquier caso, sea la estructura simple o compleja, podrán distinguirse en ella tres niveles de objetos.

- ***La raíz de la jerarquía.*** Se trata de un objeto único y especial. Este se caracteriza por estar en el nivel más alto de la estructura y suele recibir un nombre muy genérico, que indica su categoría especial, como por ejemplo objeto madre, Raíz o Entidad.

- ***Los objetos intermedios.*** Son aquellos que descienden directamente de la raíz y que a su vez tienen descendientes. Representan conjuntos o clases de objetos, que pueden ser muy generales o muy especializados, según la aplicación. Normalmente reciben nombres genéricos que denotan al conjunto de objetos que representan, por ejemplo, VENTANA, CUENTA, FICHERO. En un conjunto reciben el nombre de clases o tipos si descienden de otra clase o subclase.
- ***Los objetos terminales.*** Son todos aquellos que descienden de una clase o subclase y no tienen descendientes. Suelen llamarse casos particulares, instancias o ítems porque representan los elementos del conjunto representado por la clase o subclase a la que pertenecen.

2.2.4 Polimorfismo

Una de las características fundamentales de la OOP es el polimorfismo, que no es otra cosa que la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase a la que pertenece cada uno, con comportamientos diferentes. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes. Estos objetos recibirían el mismo mensaje global pero responderían a él de formas diferentes; por ejemplo, un mensaje "+" a un objeto ENTERO significaría suma, mientras que para un objeto STRING significaría concatenación ("pegar" strings uno seguido al otro). [17]

2.2.5 Identidad de los objetos

Las entidades conservan su identidad aunque algunas de sus propiedades cambien con el tiempo. Este concepto de identidad no se aplica a las tuplas de las bases de datos relacionales. En los sistemas relacionales, las tuplas de una relación solo se distinguen por los valores que contienen.

Los sistemas orientados a objetos proporcionan el concepto de *identificador de objeto (OID)* para identificar a los objetos. Los identificadores de objeto son únicos, es decir, cada objeto tiene un solo identificador y no hay dos objetos que tengan el mismo identificador.

Generalmente el identificador lo genera el sistema de manera automática. [13]

2.3 Clases

Generalmente, en una base de datos hay muchos objetos similares. Por similar se entiende que responden a los mismos mensajes, utilizan los mismos métodos y tienen variables del mismo nombre y del mismo tipo. Por tanto, los objetos parecidos se agrupan para formar una clase. Cada uno de estos ejemplos se denomina ejemplar de su clase.

```
class empleado {
    /* Variables */
            string  nombre;
            string  dirección;
            date    fecha_alta;
            int     sueldo;

    /* Mensajes */
            int     sueldo_anual();
            string  obtener_nombre();
            string  obtener_dirección();
            int     establecer_dirección(string nueva_dirección);
            int     antigüedad();
```

La definición muestra las variables y los mensajes a los que responden los objetos de la clase. En está definición, cada objeto de la clase *empleado* contiene las variables *nombre* y *dirección,* ambas cadenas de caracteres; *fecha_alta,* que es una fecha, y *sueldo,* que es un entero. Cada objeto responde a los cinco mensajes mostrados, llamados *sueldo_anual, obtener_nombre, obtener_dirección, establecer_dirección y antigüedad.* El nombre del tipo que precede a cada mensaje indica el tipo de la respuesta del mismo. Obsérvese que el mensaje *establecer_dirección* utiliza el parámetro *nueva_dirección que especifica el nuevo valor de la calle.*

Los métodos para el manejo de mensajes suelen definirse separados de la definición de clases. Los métodos *obtener_dirección() y establecer_dirección()* estarían definidos, por ejemplo, por el pseudocódigo:

```
string obtener_dirección(){
    return dirección;
}

int establecer_dirección(string nueva_dirección){
    dirección=nueva_dirección;
}
```

Mientras que el método *antigüedad()* se definiría:

```
int antigüedad(){
    return today() – fecha_alta;
}
```

Aquí asumimos que la función *today()* es una función que devuelve la fecha actual, y el " – " opera con ellas devolviendo el intervalo entre las dos fechas. [13]

2.4 Herencia

Los esquemas de las bases de datos orientadas a objetos suelen necesitar gran número de clases. Frecuentemente, sin embargo, varias de las clases son parecidas entre sí. Por ejemplo, supóngase que se tiene una base de datos orientada a objetos de una aplicación bancaria. Cabe esperar que la clase de los clientes del banco sea parecida a la clase de los empleados en que ambas definan variables para *nombre, dirección, etc.* Sin embargo, hay algunas variables específicas de los empleados (*sueldo,* por ejemplo) y otras específicas de los clientes (*interés_prestamo,* por ejemplo). Sería conveniente definir una representación de las variables comunes en un solo lugar. Esto solo puede hacerse si se combinan los empleados y los clientes en una sola clase.

Para permitir la representación directa de los parecidos entre las clases hay que ubicarlas en una jerarquía de especializaciones. Por ejemplo, se puede decir que *empleado* es una especialización de *persona*, dado que el conjunto de los empleados es un subconjunto de personas. Es decir, todos los empleados son personas. De manera parecida, *cliente* es una especialización de *persona.*

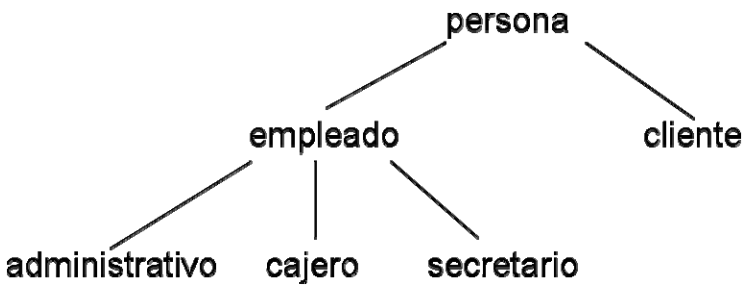


Figura 2.4.1 Jerarquía de clases.

En la Figura 2.4.1 muestra la jerarquía de clases correspondiente. Las clases mostradas en la jerarquía pueden definirse en el siguiente pseudocódigo:

```
class persona{
    string nombre;
    string dirección;
    string obtener_nombre;
    string obtener_dirección;
    int     establecer_dirección(string nueva_dirección);
};

class cliente isa persona{
    int interés_prestamo;
};
```

```
class empleado isa persona{
    date fecha_alta;
    int    sueldo;
    int    sueldo_anual();
    int    antigüedad();
};

class administrativo isa empleado{
    int número_despacho;
    int número_cuenta_corriente;
};

class cajero isa empleado{
    int horas_semana;
    int número_ventanilla;
};

class secretario isa empleado{
    int      horas_semana;
    string jefe;
};
```

La palabra clave *isa* se utiliza para indicar que una clase es una especialización de otra. La especialización de las clases de denominan *subclases*. Por ejemplo, *empleado* es una subclase de *persona*. Análogamente, *persona* es una superclase de *empleado*.

Los objetos que representan administrativos contienen todas las variables de la clase administrativo; además, los objetos que representan administrativos contienen también todas las variables de las clases *empleado* y *persona*.

Esta propiedad de que los objetos de una clase contengan las variables definidas en sus superclases se denomina *herencia* de las variables.

Los mensajes y los métodos se heredan de modo idéntico que las variables. Una ventaja importante de la herencia en los sistemas orientados a objetos es el concepto de *posibilidad de sustitución*: cualquier método de una clase dada, por ejemplo, A (o una función que utilice un objeto de la clase A como argumento) puede ser llamado de igual modo con cualquier objeto perteneciente a cualquier subclase B de A. Está característica lleva a la reutilización del código, dado que no hace falta volver a escribir los mensajes, métodos y funciones para los objetos de la clase B. [13]

2.4.1 Herencia múltiple

En la mayor parte de los casos una organización de clases con estructura de árbol resulta adecuada para describir las aplicaciones; en la organización con estructura de árbol, cada clase puede tener a lo sumo una superclase. Sin embargo, hay situaciones que no pueden representarse bien en una jerarquía de clases con estructura de árbol.

La herencia múltiple permite a las clases heredar variables y métodos de múltiples superclases. La relación entre clases y subclases se representan mediante un grafo acíclico dirigido (GAD) en el que las clases pueden tener mas de una superclase.

Por ejemplo, supóngase que los empleados pueden ser contratados por horas (para un periodo limitado) o bien a tiempo completo. Se pueden crear las subclases *por_horas* y *a_tiempo_completo* de la clase *empleado.* La subclase tendría un atributo *último_día* que especifica cuando concluye el periodo del empleo. La subclase *a_tiempo_completo* puede tener un método para el cálculo de las contribuciones al plan de pensiones de la compañía, que no es aplicable a los empleados por horas.

La clasificación expuesta de empleados como temporal y a tiempo completo es independiente de la clasificación basada en el trabajo que ellos realizan, es decir, administrativo, cajero ó secretario. Usando la herencia múltiple, simplemente se crea una nueva clase, tal como *cajero_por_horas*, que es una subclase de *por_horas* y de *cajero*. La combinación que no puede ocurrir en la vida real no necesita ser creada. La jerarquía de clases resultante aparece en la Figura 2.4.1.1. [13]

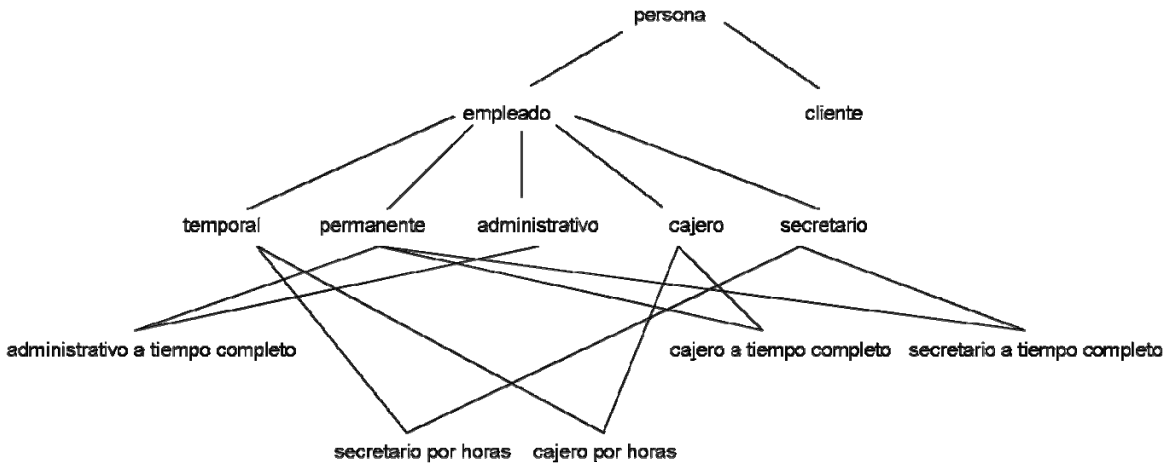


Figura 2.4.1.1 GAD de clases

2.5 Métodos

Podemos definir método como un programa procedimental o procedural escrito en cualquier lenguaje, que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes.

Son sinónimos de 'método' todos aquellos términos que se han aplicado tradicionalmente a los programas, como procedimiento, función, rutina, etc. Sin embargo, es conveniente utilizar el término 'método' para que se distingan claramente las propiedades especiales que adquiere un programa en el entorno OOP, que afectan fundamentalmente a la forma de invocarlo (únicamente a través de un mensaje) y a su campo de acción, limitado a un objeto y a sus descendientes, aunque posiblemente no a todos.

Si los métodos son programas, se deduce que podrían tener argumentos, o parámetros. Puesto que los métodos pueden heredarse de unos objetos a otros, un objeto puede disponer de un método de dos maneras diferentes:

- *Métodos propios.* Están incluidos dentro de la cápsula del objeto.
- *Métodos heredados.* Están definidos en un objeto diferente, antepasado de éste (padre,"abuelo", etc.). A veces estos métodos se llaman métodos-miembro porque el objeto los posee por el hecho de ser miembro de una clase. [17]

CAPÍTULO 3
Bases de datos relacionales orientadas a objetos (OORDB)

Los modelos de datos relacionales orientados a objetos extienden el modelo de datos relacional proporcionando un sistema de tipos más rico e incluyendo tipos de datos complejos y la programación orientada a objetos.

3.1 Modelos OORDB

Los lenguajes de consulta relacionales como SQL también necesitan ser extendidos para trabajar con el sistema de tipos enriquecido. Estas extensiones intentan conservar los fundamentos relacionales, en concreto, el acceso declarativo a los datos al tiempo que extienden la capacidad de modelado.

3.1.1 Tipos complejos

Los sistemas de tipos complejos y la programación orientada a objetos permiten que los conceptos del modelo E-R, como la identidad de las entidades, los atributos multivaluados, la generalización, la especialización, se representen directamente sin que haga falta una compleja traducción del modelo relacional.

Las extensiones de SQL que permiten los tipos complejos, incluyendo las extensiones anidadas y las características orientadas a objetos se basan en la norma SQL: 1999.

A diferencia de las bases de datos relacionales, en las bases de datos relacionales orientadas a objetos se permite que los atributos sean conjuntos, estos son ejemplares de los tipos colección.

Muchas aplicaciones actuales de las bases de datos necesitan almacenar atributos grandes, tales como la fotografía de una persona; o muy grandes, tales como imágenes médicas de alta resolución o clips de video. SQL: 1999 proporciona nuevos tipos de datos para objetos de gran tamaño:

- Para datos de caracteres (clob).
- Para datos binarios (blob).

Las letras “lob” en estos tipos de datos son acrónimos de “large object”.

3.1.2 Tipos estructurados

Los tipos estructurados se pueden declarar y usar en SQL: 1999 como en el siguiente ejemplo:

```
create type Editorial as
(
nombre varchar(20),
sucursal varchar(20)
)

create type Libro as
(
título varchar(20),
array_autores varchar(20) array[10],
fecha_pub date,
editorial editorial,
lista_palabras_clave setof(varchar(20))
)

create table Libros of type Libro.
```

La segunda instrucción define el tipo Libro, que contiene título, array_autores, que es un array de autores, una fecha de publicación, una editorial (de tipo editorial) y un conjunto de palabras clave. (La declaración de lista-palabras_clave como un conjunto usa la sintaxis extendida y no esta soportada en la norma SQL: 1999.) Los tipos ilustrados se denominan tipos estructurados en SQL: 1999.

Finalmente se crea la tabla Libros que contiene tuplas del tipo Libro. También se pueden usar tipos Fila en SQL: 1999 para definir atributos compuestos por ejemplo, se podría haber definido un atributo Editorial1 como:

```
Editorial1 row nombre (varchar(20),
                       Sucursal varchar(29))
```

En lugar de crear un tipo con nombre Editorial.

Un tipo estructurado puede tener métodos definidos sobre él. Los métodos se declaran como parte de la definición de tipos de un tipo estructurado.

```
create type Empleado as(
nombre varchar(20),
sueldo integer
)
Method incrementar(porcentaje integer)
```

El cuerpo del método se crea separadamente:

```
create method incrementar (porcentaje integer)
            for empleado
begin
    set self.sueldo = self.sueldo + (self.sueldo * porcentaje)/100
end
```

La variable self se refiere al ejemplar del tipo estructurado sobre el que se invoca el método.[13]

3.1.3 Creación de valores de tipos complejos

En SQL: 1999 se usan las funciones constructoras para crear valores de tipos estructurados. Una función con el mismo nombre que un tipo estructurado es una función constructora para el tipo estructurado. Por ejemplo, se podría declarar una constructora para el tipo Editorial como:

```
Create function Editorial(n varchar(20),s varchar(20))
Returns Editorial
Begin
    set nombre = n;
    set sucursal = s;
End
```

Se puede usar entonces Editorial (“McGraw-Hill”,”Nueva York”) para crear un valor del tipo Editorial.

En SQL: 1999, a diferencia de en las bases de datos orientadas a objetos, un constructor crea un valor del tipo, no un objeto del tipo. Es decir, el valor que crea el constructor no tiene identidad de objeto. Los objetos en SQL: 1999 se corresponden con tuplas de una relación, y se crean insertando tuplas en las relaciones.

De manera predeterminada, cada tipo estructurado tiene un constructor sin argumentos, que establece los atributos a sus valores predeterminados. Cualquiera otra constructora tiene que crearse explícitamente. Puede haber más de una constructora para el mismo tipo estructurado; aunque tengan el mismo nombre, tienen que ser distinguibles por el número de argumentos y sus tipos.

Los atributos de tipo conjunto tales como lista_palabras_clave se crean enumerando sus elementos (siguiendo a la palabra clave set). Se pueden crear valores del tipo multiconjunto al igual que con los valores de tipo conjunto, reemplazando set por multiset.

Así, se puede crear una tupla del tipo definido por la relación Libros como:

("Compiladores",array["Gómez","Santos"],Editorial("McGraw-Hill","Nueva York"),set("traducción,análisis"))

3.1.4 Herencia

La herencia puede hallarse en el nivel de los tipos o en el nivel de las tablas. En primer lugar se considerará la herencia de los tipos y después en el nivel de las tablas.

3.1.4.1 Herencia de tipos

Supóngase que se dispone de la siguiente definición de tipos para las personas:

```
create type Persona(
    nombre varchar(20),
    dirección varchar(20)
)
```

Puede que se desee guardar en la base de datos más información sobre las personas que sean estudiantes y sobre las que sean profesores. Dado que los estudiantes y los profesores también son personas, se puede utilizar la herencia para definir los tipos estudiante y profesor en SQL: 1999:

```
create type Estudiante
under Persona(
curso varchar(20),
departamento varchar(20)
)

create type Profesor
under Persona(
sueldo integer,
departamento varchar(20)
)
```

Tanto los tipos *Estudiante* como *Profesor* heredan los atributos de *Persona*, es decir, nombre y dirección. *Estudiante* y *Profesor* se denominan subtipos de *Persona* y ésta, a su vez, es un supertipo de *Estudiante* y de *Profesor*.[13]

3.1.4.2 Herencia de tablas

Las subtablas en SQL: 1999 se corresponden con la noción del modelo E-R de la especialización y generalización. Por ejemplo, supóngase que se define la tabla personas de la manera siguiente:

create table persona of Persona

Se pueden definir entonces las tablas Estudiantes y Profesores como subtablas de persona:

create table estudiantes of Estudiante
under Persona

create table profesores of Profesor
under Persona

3.2 Comparaciones entre OODB y OORDB

Las bases de datos orientadas a objetos son creadas alrededor de los lenguajes de programación persistentes y las bases de datos relacionales orientadas a objetos en cambio, son bases de datos orientadas a objetos sobre el modelo relacional.

Las extensiones persistentes de los lenguajes de programación y los sistemas relacionales orientados a objetos se han dirigido a mercados diferentes. Los sistemas relacionales orientados a objetos se dirigen a la simplificación de la realización de los modelos de datos y de las consultas mediante el uso de tipos de datos complejos. Las aplicaciones típicas incluyen el almacenamiento y la consulta de datos complejos, incluyendo los datos multimedia.

Los lenguajes declarativos como SQL, sin embargo, imponen una reducción significativa del rendimiento a ciertos tipos de aplicaciones que se ejecutan principalmente en la memoria principal y realizan gran número de accesos a la base de datos. Los lenguajes de programación persistentes se dirigen a las aplicaciones de este tipo que tienen la necesidad de elevados rendimientos. Proporcionan acceso a los datos persistentes con poca sobrecarga y eliminan la necesidad de la traducción de datos si hay que tratarlos con un lenguaje de programación. Sin embargo, son más susceptibles de deteriorar los datos debido a los errores de programación y no suelen disponer de grandes opciones de consulta.

Los puntos fuertes de los varios tipos de sistemas de bases de datos pueden resumirse de la manera siguiente:

- Sistemas relacionales: tipos de datos sencillos, lenguajes de consulta potentes, protección elevada.
- Bases de datos orientados a objetos basadas en lenguajes de programación persistentes: tipos de datos complejos, integración con los lenguajes de programación, elevado rendimiento.
- Sistemas relacionales orientados a objetos: tipos de datos complejos, lenguajes de consulta potentes, protección elevada.

Para comprender como se realiza la traducción solo es necesario examinar la forma en que algunas características del modelo E-R se traducen en relaciones. Por ejemplo, los atributos multivaluados del modelo E-R representan a los atributos de tipo conjunto del modelo relacional orientado a objetos. Los atributos compuestos representan a los tipos estructurados. Las jerarquías del modelo E-R representan a la herencia de tablas en el modelo relacional orientado a objetos. [13]

3.3 Clasificación de los sistemas manejadores de bases de datos.

En la siguiente tabla, se muestra una clasificación de las aplicaciones que requieren DBMS, además, de en que situaciones utilizar Sistemas Manejadores de Bases de Datos Relacionales (RDBMS), Sistemas Manejadores de Bases de Datos Orientados a Objetos (OODBMS) y Sistemas Manejadores de Bases de Datos Relacionales Orientados a Objetos (OORDBMS). El propósito es indicar los tipos de problemas que los DBMS pueden resolver. Como veremos, un solo DBMS no resuelve todas las aplicaciones. Entonces, indicamos porqué se espera que los problemas tratados por OORDBMS lleguen a ser cada vez más importantes en el futuro.

El esquema de clasificación utiliza una matriz de 2 x 2 que se muestra en la tabla 3.1.

Consulta a la DB	DBMS Relacional	OODBMS Relacional
Sin Consulta a la DB	Sistema de Archivos	OODBMS
	Datos Simples	**Datos Complejos**

Tabla 3.1. Una clasificación de Aplicaciones DBMS.

En la tabla, se muestran datos simples a la izquierda y datos complejos a la derecha. La complejidad de los datos puede variar generalmente entre estos dos extremos en una manera continua. Similarmente, en el eje vertical se distingue si se requiere capacidad de consulta.

Se puede examinar el uso de los DBMS y después ponerlo en uno de los cuatro cuadros en la tabla 3.1 dependiendo de sus características. Se mostrarán los requisitos que cada DBMS tiene y entonces sugerir que hay una opción natural para cada uso. [21]

3.3.1 Datos simples sin consulta a la DB

El DBMS obvio para usar en el cuadro inferior izquierdo de nuestra matriz es el sistema de ficheros proporcionado por el vendedor del sistema operativo. De hecho, todos los editores de textos usan este nivel primitivo del servicio del DBMS actualmente, y no se requiere algo sofisticado. La razón es absolutamente simple: si no tienes ninguna necesidad de consultas y ninguna necesidad de datos complejos, entonces el servicio proporcionado por el sistema de ficheros es perfectamente adecuado. Por otra parte, el sistema de archivos tiene invariablemente rendimiento más alto que el sistema más sofisticado. Por lo tanto, si no necesitas el servicio, entonces no hay necesidad de pagar por él.

Si se cuenta con una aplicación como la anterior, lo adecuado es utilizar un sistema de archivos proporcionado por tu equipo de cómputo. Por lo tanto, la esquina inferior izquierda corresponde al "Sistema de archivos".[21]

3.3.2 Datos simples y consulta a la DB

Se ilustrará la esquina superior izquierda con un ejemplo bien conocido. Suponer que deseamos almacenar la información sobre cada empleado en una compañía. Se desea registrar el nombre del empleado, edad, sueldo, y departamento. Además, se requiere la información sobre los departamentos: nombre del departamento, presupuesto y número de piso en que se localiza. El esquema para esta información se puede capturar por las declaraciones SQL siguientes:

```
create table empleado (
  nombre    varchar(30),
  edad      int,
  salario   float,
  dept      varchar(20));
```

```
create table departamento (
  dnombre   varchar(20),
  presupuesto  float,
  piso   int);
```

Se pueden hacer algunas consultas, por ejemplo:

Encontrar los empleados que tengan menos de 40 años y ganen más de $40,000.

```
select nombre
from empleado
where edad < 40 and salario > 40000;
```

Mostrar los nombres de los empleados que trabajan en el primer piso.

```
select nombre
from empleado
where departamento  in
  (select dnombre
  from departamento
  where piso = 1);
```

Las consultas que el usuario desea saber son simples, expresadas en estándar SQL 92. Estas aplicaciones tienen los siguientes requisitos:

Lenguaje de Consulta: Puede utilizar SQL-89. Se prefiere SQL-92.

Herramientas del Cliente: Se requieren herramientas que permitan al programador la entrada y salida de datos. Estos son llamados Lenguajes de Cuarta Generación (4GLs). Además estas herramientas deben incluir: generación de reportes, herramientas de diseño de bases de datos y la capacidad de llamar servicios del DBMS.

Funcionamiento: Se requiere procesamiento de transacciones. Esto es, muchos clientes requieren servicios del DBMS de terminales, clientes o PCs (normalmente son transacciones simples, mas frecuentemente actualizaciones). Cuando existen transacciones paralelas que puedan causar conflictos, se desea que el resultado sea fiable. Además, hay un requisito absoluto, nunca perder los datos del usuario, sin importar que tipo de problema haya ocurrido.

Seguridad: Puesto que los usuarios ponen datos sensibles, tales como sueldos, hay un requisito riguroso: la seguridad del DBMS. Consecuentemente, el DBMS debe funcionar en un espacio separado del uso del cliente, de modo que ocurra una travesía siempre que se ejecute una petición del servicio del DBMS. De esta manera, el DBMS puede funcionar con un identificador de usuario diferente al utilizado para otro uso. Por otra parte, los ficheros utilizados en la base de datos son legibles y almacenados solamente por el DBMS. Esta arquitectura cliente-servidor, se muestra en la figura 3.2.

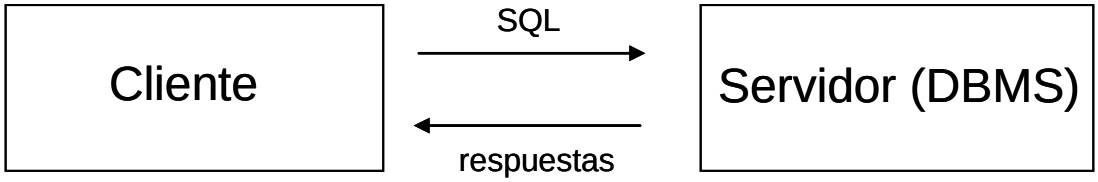


Figura 3.2. Arquitectura Estándar Cliente-Servidor

En todo caso, el usuario requiere un DBMS optimizado para el tratamiento transaccional atendiendo a muchos usuarios simultáneos con comandos simples.

En conclusión, si se tiene una aplicación del tipo utilizado en la tabla 3.2, use un DBMS Relacional. [21]

3.3.3 Datos complejos sin Consulta a la DB

Suponer que el usuario es el planificador de las instalaciones para una compañía en la cual nadie tiene una oficina. Por lo tanto, organizan a todos los empleados en cubículos, con particiones separadas. Una compañía ejemplo de esto es Hewlett-Packard. En tal compañía, los departamentos crecen y decrecen, además de que contratan y despiden empleados. En un cierto plazo, el conjunto de empleados en el espacio físico del edificio llega a ser insuficiente o desaprovechado, y se autoriza un nuevo cambio global. Este reordenamiento del espacio libre y de los empleados es el próximo ejemplo.

La base de datos para esta aplicación puede ser expresada de la forma siguiente:

```
create table empleado (
  nombre       varchar(30),
  espacio      polígono,
  adyacencia  junto-a (empleado));

create table pisos (
  número     int,
  pep        polígono);
```

Para cada empleado deseamos registrar su nombre, el espacio que ocupa, así como el conjunto de los empleados con quienes él comparte una pared en común (adyacencia). Para cada piso necesitamos registrar el número y los pies cuadrados asignables (pep) del piso. Esta cantidad es el contorno del edificio menos los cuartos del resto, los ejes del elevador y las salidas de auxilio. Como tal, es un polígono con "agujeros". Asimismo, el espacio es un polígono y la adyacencia es un conjunto. Esta aplicación se puede escribir en pseudocódigo como sigue:

```
main ()
{
read all empleados;
read all pisos;
compact();
write all empleados;
}
```

Aquí, el programa debe leer todos los registros del empleado para comprobar su espacio actual así como todos los pisos para conseguir su pep. El programa construirá una cierta clase de estructura de la memoria virtual para el paso siguiente del programa. Esta rutina de la compactación entonces ejecutará esta estructura, quizás muchas veces, de generar una asignación de empleados en el pep en cada piso. Cuando la compactación esta completa, cada registro del empleado se debe reescribir con una nueva asignación del espacio.

Obviamente, este programa lee ambos conjuntos de datos, los calcula, y después escribe un nuevo conjunto. Por lo tanto, es análogo a un editor de textos que lee y después escribe un solo archivo. Pero a diferencia del editor de textos, los datos implicados son algo complejos. Como tal, se clasifica en la esquina inferior derecha de la tabla 1.

Usar un sistema de ficheros tradicional para este uso es bastante costoso. La aplicación debe leer la información de los empleados y los pisos. Una solución mucho mejor sería apoyarse en el almacenaje persistente para el lenguaje de programación en el cual se escribe. Para esto, podemos utilizar C++.

Un lenguaje de programación persistente ofrece el mejor soporte para esta aplicación de compactación.

```
main ()
{
read all empleados;;
read all pisos;;
compact();
write all empleados;
}
```

A simplemente tener que escribir:

```
main ()
{
compact();
}
```

Note que esta aplicación tiene los siguientes requisitos:

Lenguaje de consulta: no se requiere ninguno para esta aplicación. Si uno está presente, no responde a ningún propósito útil.

Seguridad: El requisito de funcionamiento antes dicho está en la raíz de la arquitectura de los lenguajes persistentes. Específicamente, suponer que el usuario ejecuta el comando siguiente:

```
J = J + 1;
```

Es decir, se incrementa J. Si J no es persistente, entonces esta sentencia se ejecutaría en 1 microsegundo o menos. Por otra parte, si J es persistente, entonces esta sentencia realiza una actualización. Si el sistema de almacenamiento se ejecuta en un espacio de dirección diferente al del programa del usuario, entonces una interrupción del espacio de dirección ocurre al procesar este comando. Consecuentemente, los comandos se ejecutarán quizás 2 o 3 veces más lentos que en el caso no persistente. Este funcionamiento es inaceptable a los usuarios, y esto hace a los diseñadores de los sistemas persistentes de almacenaje ejecutarlos en el mismo espacio de dirección que el programa de usuario.

Figura 3.3. La Arquitectura de los Lenguajes Persistentes

Si se tiene una aplicación con características como la de la esquina inferior derecha, se debe elegir un OODBMS con un lenguaje persistente.

¿Qué sucedería si tengo un problema como el de la esquina superior izquierda de la tabla 1 y lo solucionó con un OODBMS? Una vez más el resultado será costoso. Específicamente, la mayoría de los OODBMS tienen SQL muy limitado, y varios no apoyan las actualizaciones en SQL. Como tal, el usuario tendrá que utilizar C++ para expresar sus transacciones, dando por resultado mucho más código. Además, los productos de los OODBMS no están optimizados para soportar 50 o 100 actualizaciones concurrentes, y tienden a ofrecer poco rendimiento en este campo. Es decir, usar un DBMS diseñado para un tipo de aplicación en un ambiente diferente daría lugar a un desastre. El resultado es que los OODBMS no trabajan bien en problemas como el de la esquina superior izquierda. [21]

3.3.4 Datos complejos y consulta a la DB

El departamento del estado de California de los recursos de agua (DWR) se encarga del manejo de la mayor parte de de los canales y de la irrigación en California, así como de una colección de acueductos, incluyendo el proyecto del agua del estado. Para documentar sus instalaciones, mantienen una biblioteca de fotografías de 35mm. En un cierto plazo esta biblioteca ha crecido a 500.000 fotografías y es accedida actualmente entre 5 y 10 veces al día por los empleados de DWR y otros.

El cliente está solicitando una forma de ordenar las fotografías por su contenido. Por ejemplo, un empleado tiene que dar una presentación en las cual necesita una foto panorámica, para una estación de bombeo masiva que lleve el agua al norte de California sobre las montañas de Tecahappi en California meridional. Otra petición, fue para una foto de la bahía de San Francisco en la puesta del sol, mientras que un tercero pudo solicitar una foto de un depósito donde el nivel del agua era muy bajo.

DWR ha encontrado que es muy difícil organizar las fotografías. Poner en un índice todas las diapositivas según una colección predefinida de conceptos es un trabajo muy costoso. Por otra parte, los clientes están interesados en conocer los cambios en un cierto plazo. Por ejemplo, el agua baja en un depósito nunca interesaba, hasta que la sequía actual comenzó hace 7 años. Además, las especies en peligro de extinción son un problema. Actualmente, tienen un título escrito sobre cada diapositiva. Por ejemplo:

Foto de la presa Auburn tomada durante la construcción del andamio.

Se utiliza un sistema que identifica las palabras claves de las fotografías. Este sistema no está funcionando muy bien, porque muchos conceptos de interés no se mencionan en el título.

Por consiguiente, DWR está digitalizando todas las fotografías y está en curso de construir la siguiente base de datos:

```
create table fotografía (
  id       int,
  fecha    date,
  título   document,
  foto   photo_CD_image);

create table regiones(
  nombre     varchar(30),
  localización  point);
```

Cada fotografía tiene un identificador, la fecha en que fue tomada, el título, y fue digitalizada en formato Kodak Photo-CD. De hecho, el formato del Photo-CD es una colección de 5 imágenes con un tamaño de 128 de x 192 pixeles, con una máxima imagen de color de 2K x 3K. Actualmente, DWR ha digitalizado cerca de 20,000 imágenes y tienen planeado construir una base de datos que tendrá un tamaño alrededor de 3 Terabytes.

DWR está muy interesado en clasificar sus imágenes electrónicamente. Según lo observado arriba, clasificarlas es inverosímil. Una de las cualidades que se desean capturar es la localización geográfica de cada diapositiva. Su técnica para lograr este registro geográfico implica una base de datos espacial de dominio público sobre el espacio geológico de US. Específicamente, tienen los nombres de todas las regiones que aparecen en cualquier mapa topográfico de California junto con la localización del mapa de la región. Entonces, el propósito es examinar el título de cada fotografía y comprobar si corresponde con el nombre de una región. Si es así la localización de la región es una buena clave para la posición geográfica de la diapositiva.

Además, DWR está interesado en los programas que examinan una imagen y comprueban las cualidades de esta. De hecho, uno puede encontrar una puesta del sol en esta biblioteca en particular buscando el color naranja en la parte alta de la foto. El agua baja en un depósito exige buscar un objeto azul rodeado por un anillo marrón. Muchas cualidades de una foto de las cuales DWR tenga algún interés, se pueden encontrar al usar técnicas de concordancia con el modelo deseado. Por supuesto, algunas cualidades son mucho más difíciles, por ejemplo comprobar si la foto contiene una especie en peligro de extinción. Estas cualidades más duras tendrán que esperar a avances futuros en el reconocimiento de patrones.

Por ejemplo, para encontrar una foto de la puesta del sol tomada a 20 millas de Sacramento, los clientes desean una interfaz amistosa que asista la consulta siguiente de SQL:

```
select id
from fotografía P, regiones L S
where Puesta_sol (P.foto) and
contains (P.título, L.nombre) and
L.localización |20| S.localización and
S.nombre = 'Sacramento';
```

Esta consulta se puede explicar comenzando por el final. Primero, necesitamos encontrar Sacramento en la tabla de regiones. Esto implica la localización geográfica de Sacramento (que esta en varios mapas topográficos). Entonces, encontramos otras regiones (L.localización) que esta a 20 millas de S.localización. |20| es un operador definido por el usuario definido por dos operandos, de tipo puntero, que devuelve true si los dos punteros están a menos de 20 millas uno del otro. Esto es un conjunto de regiones que podemos utilizar para comprobar si aparece en un título de alguna foto. Su contenido es una función definida por el usuario la cual acepta dos argumentos, un documento y una palabra clave, y comprueba si la palabra clave aparece en el documento. Esto muestra el conjunto de fotos que son candidatas al resultado de la consulta. Antes, la puesta del sol es una segunda función definida por el usuario que examina los bits en una imagen para buscar si tienen color naranja en la parte alta. El resultado es el deseado por el cliente.

Obviamente, esta aplicación exige muchas consultas con datos complejos. Como tal, es un ejemplo de una aplicación de la esquina superior derecha.

Lenguaje de consulta: Observe que la consulta de ejemplo sobre las puestas del sol en Sacramento, tiene cuatro cláusulas. La primera contiene una función definida por el usuario, Puesta_sol, y no está en modo de SQL-92. La segunda cláusula contiene otra función definida por el usuario. La tercera cláusula contiene un operador definido por el usuario, |20|, el cual no está en SQL-92. Como tal, se requiere un lenguaje de consulta que permita funciones definidas por el usuario y operadores. La primera versión estándar del SQL con estas capacidades es SQL-3. Por lo tanto, el usuario requiere un DBMS SQL-3. Cualquier DBMS SQL-2 es esencialmente inútil en este uso, porque tres de las cuatro cláusulas no se pueden expresar en SQL-2.

Herramientas: Se desea programar la aplicación del usuario exhibiendo en la pantalla un mapa del estado de California. Entonces, el usuario podría explorar con un puntero el área del estado que le interesa. Como resultado, el usuario le gustaría ver un mapa del condado de Sacramento con un conjunto de imágenes colocadas en su localización geográfica. Con un filtro sobre el condado, él podría examinar la imagen que le interesa. Además, el usuario entonces quisiera aplicar un zoom en determinadas áreas, y se mostrarían una serie de imágenes de alta resolución almacenadas como objetos Photo-CD. Tal interfaz "pan and zoom" es típica de productos científicos de la visualización tales como Khoros, explorer y AVS. Como tal, el usuario quisiera un sistema de la visualización integrado cuidadosamente con su DBMS. Observe que un 4GL estándar es casi inútil para esta aplicación; no hay una herramienta comercial en vista.

Rendimiento: El usuario requiere que las consultas tales como la puesta del sol de Sacramento se realicen muy rápidamente. Para realizarse bien en este ambiente, se requiere un conjunto de optimizaciones. Por ejemplo, la función Puesta_sol examinará típicamente más de 100 millones de instrucciones. Por tanto, el optimizador de la consulta tendría una cláusula de la forma:

where sunset (image) and date < "Jan 1, 1985"

Los métodos de acceso tradicionales (los B-trees y hashing) son inoperantes para este tipo de cláusulas. Acelerar tales cláusulas espaciales, requiere un método de acceso espacial, tal como un archivo grid, un R-tree, o un K-D-B-tree. Un DBMS debe tener tales “métodos de objetos específicos”, o debe permitir que un usuario o un programador experto del sistema, agregue un nuevo método. Obviamente, la mejor respuesta es la técnica de la última opción.

Seguridad: Hemos observado ya que SQL-92 no resuelve el problema anterior, porque SQL-92 no puede expresar las consultas del usuario. Consecuentemente, deben ser ejecutadas en un programa del usuario, de tal modo se genera mucho trabajo para el usuario. Además, si la función Puesta_sol es ejecutada por el usuario, entonces una imagen muy grande será transmitida sobre una conexión del servidor del cliente. Esto causará un problema de funcionamiento severo. Como tal, un DBMS relacional será muy costoso y lento.

Similarmente, un lenguaje persistente no tiene un lenguaje de consulta capaz de expresar la consulta del usuario. Por lo tanto, él debe expresar su consulta como un programa C ++, conduciendo a costos y sufrimiento significativos. [21]

CAPÍTULO 4
PostgreSQL

Postgres95 desarrollado originalmente en el Departamento de Ciencias de Computación de la Universidad de California en Berkeley, liderado por el profesor Michael Stonebraker, es la base del Sistema PostgreSQL.

4.1 PostgreSQL

PostgreSQL es un ORDBMS con licencia BSD (esta licencia básicamente consiste en que ves el código, puedes redistribuirlo y puedes modificarlo. La Free Software Fundation (FSF) lo considera junto con la licencia General Public Licence (GPL) de software libre como ampliamente utilizado. Este tipo de software es fundamental en cualquier esquema computacional moderno. [23]

PostgreSQL incorpora cuatro conceptos adicionales básicos:

- Clases.
- Herencia.
- Tipos.
- Funciones.

Otras características que aportan potencia y flexibilidad son:

- Restricciones (Constraints).
- Disparadores (Triggers).
- Reglas (Rules).
- Integridad transaccional.

Estas características colocan a Postgres en la categoría de las Bases de Datos identificadas como objeto-relacionales.

PostgreSQL es libre y está disponible todo su código fuente.

En general, cualquier plataforma moderna compatible con Unix debería ser capaz de ejecutar PostgreSQL, además de ejecutarse nativamente sobre sistemas operativos basados en Microsoft Windows NT tales como Win2000, WinXP y Win2003. Existe incluso un port para Novell Netware 6 y una versión para OS/2 (eComStation).

En este proyecto de investigación se utiliza la versión 8.1.3-1 de PostgreSQL.

PostgreSQL tiene la mayoría de las características presentes en los grandes DBMSs comerciales, tales como transacciones, subconsultas, disparadores (triggers), vistas, integridad referencial con llaves externas, y bloqueo sofisticado. También tiene algunas características que no tienen las otras, como tipos definidos por el usuario, herencia, reglas, y control de concurrencia multi-versión para reducir el bloqueo de transacciones.

El desempeño de PostgreSQL es comparable con el de otras bases de datos comerciales y de código abierto. Su desempeño es usualmente +/-10% comparado con otras bases de datos.

Un DBMS debe ser fiable, o es inútil. Cada versión tiene al menos un mes de pruebas beta, y la historia de liberaciones muestra que se proveen versiones estables y sólidas que se encuentran listas para su uso en producción.

La instalación de PostgreSQL incluye solamente las interfaces de C y C ++. Todas las demás interfaces son proyectos independientes que son descargados por separado; el estar separados les permite tener su propia agenda de liberación y equipos de desarrollo.

Algunos otros lenguajes de programación como PHP incluyen una interfaz para PostgreSQL. Las interfaces para lenguajes como Perl, TCL, Python y muchas otras están disponibles en http://gborg.postgresql.org en la sección de Controladores e Interfaces y por búsqueda en Internet.

Por omisión, PostgreSQL sólo permite conexiones desde el equipo local utilizando sockets de dominio Unix o conexiones TCP/IP. Otros equipos no podrán conectarse a menos que se modifique listen_addresses en el archivo postgresql.conf, se habilite la autenticación basada en anfitrión modificando el archivo $PGDATA/pg_hba.conf y se reinicie el servidor.

A diferencia de la mayoría de otros sistemas de bases de datos que usan bloqueos para el control de concurrencia, Postgres mantiene la consistencia de los datos con un modelo multiversión (MVCC). Esto significa que mientras se consulta una base de datos, cada transacción ve una imagen de los datos (una versión de la base de datos), sin tener en cuenta el estado actual de los datos. Esto evita que la transacción vea datos inconsistentes que pueden ser causados por la actualización de otra transacción concurrente en la misma fila de datos, proporcionando aislamiento transaccional para cada sesión de la base de datos.

La principal diferencia entre multiversión y el modelo de bloqueo es que los modelos MVCC derivados de una consulta (lectura) de datos no entran en conflicto con los bloqueos derivados de la escritura de datos y de este modo la lectura nunca bloquea la escritura y viceversa.

Pgadmin es una herramienta de propósito general para diseñar, mantener y administrar las bases de datos de Postgres. Funciona bajo Win 95/98 y NT.

Características incluidas:

- Entradas SQL aleatorias.
- Pantallas de información y ayudas para bases de datos, tablas, índices, secuencias, vistas, programas de arranque, funciones y lenguajes.
- Preguntas y respuestas para configurar usuarios, grupos y privilegios.
- Control de revisión con mejora de la generación de scripts.
- Configuración de las tablas de Microsoft MSysConf.
- Ayudas para importar y exportar datos.
- Ayudas para migrar bases de datos.
- Informes predefinidos en bases de datos, tablas, índices, lenguajes, secuencias y vistas.

Pgadmin se distribuye separadamente de Postgres y puede ser descargado desde la dirección http://www.pgadmin.freeserve.co.uk. [19]. En la versión 8.1.3 ya viene incorporado a Postgres y es la versión III.

La seguridad de la base de datos esta implementada en varios niveles:

- Protección de los ficheros de la base de datos. Todos los ficheros de la base de datos están protegidos contra escritura por cualquier cuenta que no sea la del superusuario de Postgres.
- Las conexiones de los clientes al servidor de la base de datos están permitidas, por defecto, únicamente mediante sockets Unix locales y no mediante sockets TCP/IP. Ha de arrancarse el demonio con la opción –i para permitir la conexión de clientes no locales.
- Las conexiones de los clientes pueden ser autentificados mediante otros paquetes externos.
- A cada usuario de Postgres se les asigna un nombre de usuario y (opcionalmente) una contraseña. Por defecto, los usuarios no tienen permiso de escritura a bases de datos que no hayan creado.
- Los usuarios pueden ser creados en grupos, y el acceso a las tablas puede restringirse en base a esos grupos. [20]

4.2 Otras características de PostgreSQL

Los límites para una base de datos desarrollada en PostgreSQL en general son:

¿Tamaño máximo para una base de datos? Ilimitado (existen bases de datos de 32 TB, dependiendo de).

¿Tamaño máximo para una tabla? 32 TB.
¿Tamaño máximo para un registro? 1.6 TB.

¿Tamaño máximo para un campo? 1 GB.
¿Número máximo de registros en una tabla? Ilimitado.
¿Número máximo de columnas en una tabla? 250-1600 dependiendo del tipo de columna.
¿Número máximo de índices en una tabla? Ilimitado.

Obviamente no son ilimitados, ya que están limitados al espacio disponible en disco y al espacio de memoria swap. El desempeño disminuirá cuando estos valores se vuelvan inusualmente largos.

Una limitante es que los índices no pueden ser creados en columnas con más de 2,000 caracteres. Afortunadamente dichos índices son raramente necesarios.

Los valores NULOS son almacenados como mapas de bits, así que utilizan poco espacio.

Cada registro que es creado en PostgreSQL obtiene un OID único a menos que sea creado con el comando WITHOUT OIDS. Los OID son enteros únicos de 4 bytes asignados automáticamente que son únicos a lo largo de toda la instalación. Sin embargo, si estos son mayores a 4 billones, entonces los OID empiezan a duplicarse. PostgreSQL utiliza los OID para vincular tablas de sistema internas.

Para enumerar las columnas en las tablas de usuario de manera única, es mejor utilizar SERIAL en lugar de OID debido a que las secuencias SERIAL son únicas dentro de una tabla. Este es el equivalente a auto_increment en mysql y a identity en sql server. Puede utilizar SERIAL8 para almacenar secuencias de valores de 8 bytes. [22]

4.3 Implantación de una OORDB experimental

Se implantará una OORDB experimental para desarrollar los métodos necesarios para el tratamiento a las restricciones de integridad.

4.3.1 Descripción del Problema

Una compañía de alquiler de coches desea tener una base de datos de todos los vehículos de su flota actual. Para todos los vehículos se requiere su número de bastidor, matrícula, fabricante, modelo, fecha de adquisición y su color. Se incluyen datos específicos para algunos tipos de vehículos.

- Camiones: capacidad de carga.
- Coches deportivos: potencia, edad mínima del arrendatario.
- Compactos: Número de plazas.
- Camionetas: altura de los bajos, eje motor (tracción a dos ruedas o a las cuatro).

Un arrendatario puede rentar hasta 2 vehículos en cada ocasión y debe cumplir con los siguientes requisitos:

- Ser mayor de 23 años.
- Licencia de conducir vigente.
- Tener tarjeta de crédito bancaria.
- Comprobante de domicilio.
- Identificación oficial (credencial de elector, cartilla militar, pasaporte).

Se desea guardar los datos del arrendatario:

- Nombre.
- Compañía.
- Email.
- Teléfono.
- Fax (opcional).
- Dirección.
- Ciudad.
- Estado.
- País.
- C.P.

En caso de que el pago de la renta sea con tarjeta de crédito bancaria, se desean almacenar los siguientes datos:

- Tipo de Tarjeta de Crédito.
- Nombre que aparece en la tarjeta de crédito.
- Número de la tarjeta de crédito.
- Código de verificación (3 dígitos).
- Fecha de vencimiento.

4.3.2 Diagrama E – R

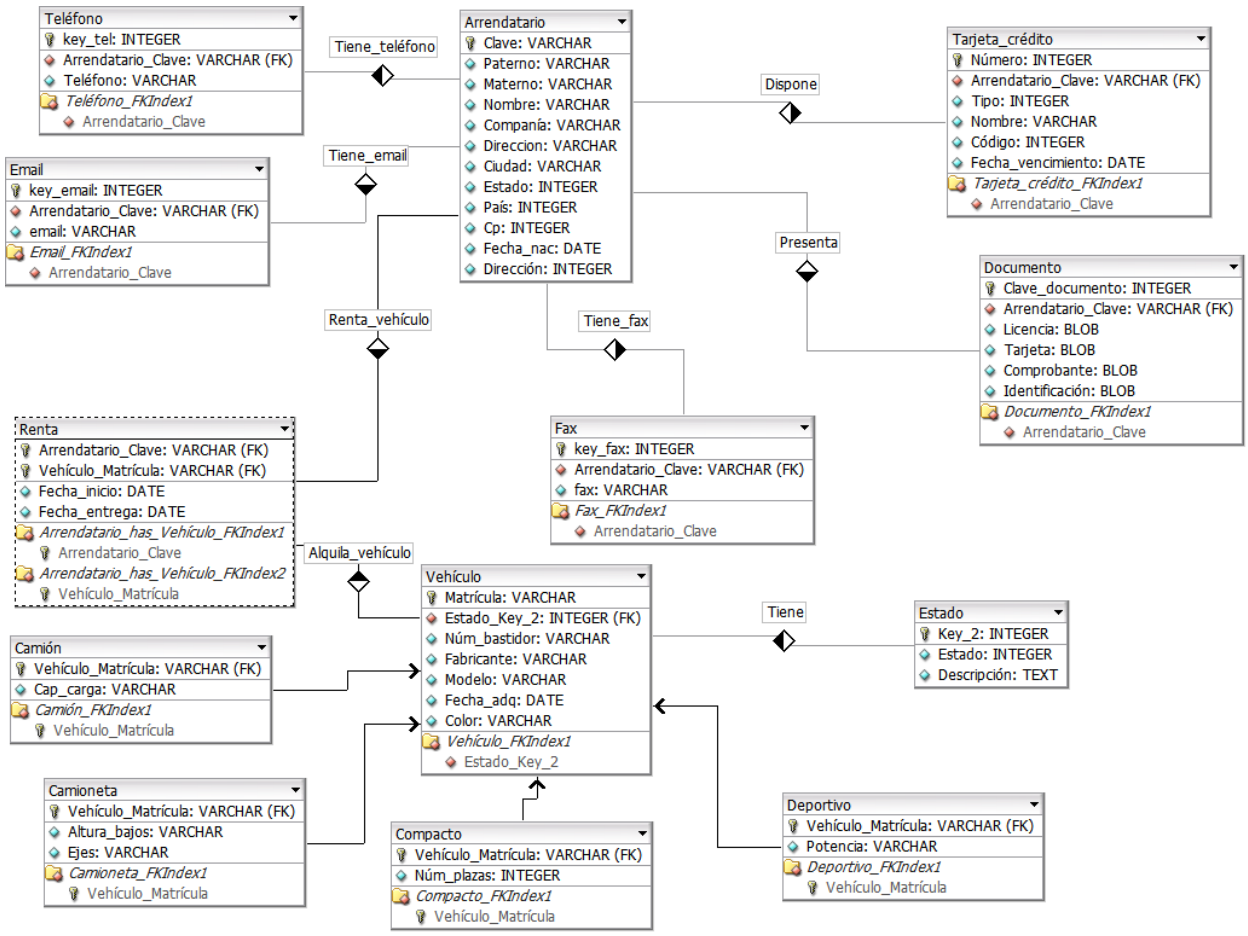
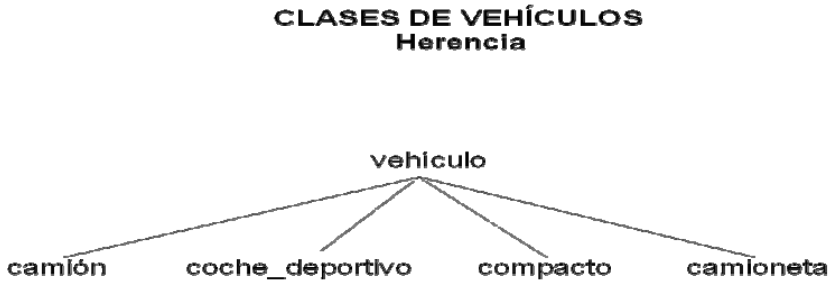


Figura 4.1 Diagrama E-R

4.3.3 Grafo acíclico dirigido



GAD

Figura 4.2 Grafo Acíclico Dirigido

4.3.4 Diccionario de datos

Entidades

- **VEHÍCULO:** Se refiere al conjunto de vehículos que pueden rentarse.
- **CAMIÓN:** Se refiere al conjunto de vehículos con un peso mayor a 8 toneladas.
- **DEPORTIVO:** Se refiere al conjunto de vehículos que pueden alcanzar gran velocidad en pocos segundos.
- **COMPACTO:** Se refiere al conjunto de vehículos pequeños, de 2 o 4 puertas.
- **CAMIONETA:** Se refiere al conjunto de vehículos destinado al transporte y carga.
- **ARRENDATARIO:** Se refiere al conjunto de personas que pueden rentar un vehículo.
- **TARJETA _ CRÉDITO:** Se refiere al conjunto de tarjetas de crédito bancarias.
- **ESTADO:** Se refiere al conjunto de estados de funcionamiento de un vehículo.
- **DOCUMENTO:** Se refiere al conjunto de documentos presentados por el arrendatario.
- **TELÉFONO:** Se refiere al conjunto de números telefónicos del Arrendatario.
- **EMAIL:** Se refiere al conjunto de direcciones de correo electrónico del Arrendatario.
- **FAX:** Se refiere al conjunto de números telefónicos a los cuales se puede enviar un fax.

Relaciones

- **DISPONE:** Relaciona a las entidades ARRENDATARIO Y TARJETA_CRÉDITO.
- **PRESENTA:** Relaciona a las entidades ARRENDATARIO Y DOCUMENTO.
- **RENTA:** Relaciona a las entidades ARRENDATARIO Y VEHÍCULO.
- **TIENE:** Relaciona a las entidades VEHÍCULO Y ESTADO.
- **ES:** Es una generalización de la entidad VEHÍCULO a las entidades CAMIÓN, CAMIONETA, COMPACTO Y DEPORTIVO. (Herencia de atributos).
- **TIENE_TELÉFONO:** Relaciona a las entidades ARRENDATARIO Y TELÉFONO.
- **TIENE_FAX:** Relaciona a las entidades ARRENDATARIO Y FAX.
- **TIENE_EMAIL:** Relaciona a las entidades ARRENDATARIO Y EMAIL.

TABLA DE RELACIONES

Relaciones	Cardinalidad	Atributos	Participación total	Participación opcional
DISPONE	Uno a varios.	No	Si	Si
PRESENTA	Uno a varios.	No	Si	No
RENTA	Varios a varios	Si	No	Si
TIENE	Uno a varios.	No	Si	No
TIENE_TELÉFONO	Uno a varios.	No	Si	No
TIENE_FAX	Uno a varios.	No	Si	No
TIENE_EMAIL	Uno a varios.	No	Si	No

Tabla 4.3 Tabla de Relaciones

TABLA DE ENTIDADES

Entidades	Atributos	Descripción
Arrendatario		
	CLAVE	Identificador de arrendatario.
	PATERNO	Apellido Paterno del arrendatario.
	MATERNO	Apellido Materno del arrendatario.
	NOMBRES	Nombre(s) de pila del arrendatario.
	COMPANIA	Fecha de ingreso del arrendatario.
	DIRECCIÓN	Dirección del arrendatario.
	CIUDAD	Ciudad donde radica el arrendatario.
	ESTADO	Estado donde radica el arrendatario.
	PAÍS	País donde radica el arrendatario
	CP	Código Postal donde radica el arrendatario
	FECHA_NAC	Fecha de nacimiento del arrendatario
Tarjeta_crédito		
	NUMERO	Número impreso en la tarjeta de crédito.
	TIPO	Tipo de tarjeta de crédito (visa,mastercard)
	NOMBRE	Nombre del titular de la tarjeta impreso en la tarjeta de crédito
	CÓDIGO	Código de seguridad de la tarjeta de crédito (3 dígitos).
	FECHA_VEN	Fecha de vencimiento de la tarjeta de crédito.
Documento	*LICENCIA*	Imagen de la licencia de conducir.
	TARJETA	Imagen(s) de la(s) tarjeta(s) de crédito
	COMPROBANTE	Imagen del comprobante domiciliario.
	IDENTIFICACIÓN	Imagen de la identificación oficial.
Estado	*KEY*	Identificador del estado del vehículo
	ESTADO	Estado físico del vehículo
	DESCRIPCIÓN	Descripción del estado físico del vehículo.
Vehículo	*MATRÍCULA*	Número de placas del vehículo.
	NÚM_BASTIDOR	Número del bastidor de la carrocería.
	FABRICANTE	Fabricante o marca del vehículo.
	MODELO	Modelo del vehículo.
	FECHA_ADQ	Fecha de compra del vehículo
	COLOR	Color del vehículo.
Camión	*CAP_CARGA*	Capacidad de carga del camión.
Camioneta	*ALTURA_BAJOS*	Altura de los bajos de la camioneta.
	EJES	Tracción de dos ruedas o cuatro.
Compacto	*NÚM_PLAZAS*	Capacidad del vehículo (número de personas).
Deportivo	*POTENCIA*	Potencia del motor del vehículo.
Teléfono	*KEY_TEL*	Identificador del número telefónico
	TELÉFONO	Número telefónico
Fax	*KEY_FAX*	Identificador del número de fax
	FAX	Número de fax
Email	*KEY_EMAIL*	Identificador de correo electrónico
	EMAIL	Dirección de correo electrónico

Tabla 4.4 Tabla de Entidades

4.3.5 Modelo relacional

TABLAS

Nombre de la tabla	Columnas
Arrendatario	
	CLAVE ***(PK)***
	PATERNO
	MATERNO
	NOMBRES
	COMPANIA
	EMAIL
	TELÉFONO
	FAX
	DIRECCIÓN
	CIUDAD
	ESTADO
	PAÍS
	CP
	FECHA_NAC
	MATRÍCULA_VEH ***(FK)***

Nombre de la tabla	Columnas
Tarjeta_crédito	
	NÚMERO ***(PK)***
	TIPO
	NOMBRE
	CÓDIGO
	FECHA_VEN
	CLAVE_ARC ***(FK)***

Nombre de la tabla	Columnas
Documento	
	KEY_DOC ***(PK)***
	LICENCIA
	TARJETA
	COMPROBANTE
	IDENTIFICACIÓN
	CLAVE_ARD ***(FK)***

Nombre de la tabla	Columnas
Estado	
	KEY_ESTADO ***(PK)***
	ESTADO
	DESCRIPCIÓN

Nombre de la tabla	Columnas
Vehículo	
	MATRÍCULA ***(PK)***
	NÚM_BASTIDOR
	FABRICANTE
	MODELO
	FECHA_ADQ
	COLOR
	KEY_ESTADO ***(FK)***

Nombre de la tabla	Columnas
Camión	
	CAP_CARGA

Nombre de la tabla	Columnas
Camioneta	
	ALTURA_BAJOS
	EJES

Nombre de la tabla	Columnas
Compacto	
	NÚM_PLAZAS

Nombre de la tabla	Columnas
Deportivo	
	POTENCIA

Tabla 4.5 Modelo Relacional

4.3.6 Normalizado de tablas

Por la primera forma normal de Codd se crearon 3 tablas adicionales para cumplir con la regla de dominios atómicos. De la tabla Arrendatario se quitaron los atributos teléfono, email y fax. Quedando de la siguiente manera:

Nombre de la tabla	Columnas
Arrendatario	
	CLAVE ***(PK)***
	PATERNO
	MATERNO
	NOMBRES
	COMPANIA
	DIRECCIÓN
	CIUDAD
	ESTADO
	PAÍS
	CP
	FECHA_NAC
	MATRÍCULA_VEH ***(FK)***

Nombre de la tabla	Columnas
Teléfono	
	KEY_TEL ***(PK)***
	TELÉFONO
	CLAVE_ART ***(FK)***

Nombre de la tabla	Columnas
Email	
	KEY_EMAIL ***(PK)***
	EMAIL
	CLAVE_ARE ***(FK)***

Nombre de la tabla	Columnas
Fax	
	KEY_FAX ***(PK)***
	FAX
	CLAVE_ARF ***(FK)***

Tabla 4.6 Normalizado de tablas

El esquema cumple con la segunda y tercera forma normal, ya que todos los atributos dependen totalmente de la llave primaria y no existen transitividades entre los atributos.

CAPÍTULO 5
Tratamiento de Restricciones de Integridad en POSTGRESQL

El presente capítulo muestra el soporte de integridad y el resultado de la evaluación de integridad realizada a este manejador.

5.1 Resultados de la evaluación de integridad

Para el desarrollo de la evaluación de integridad se utilizó la base de datos experimental **"Autos"** implementada en PostgreSQL versión 8.1.3.1 y pgAdmin III versión 1.4.1 que es una aplicación gráfica para gestionar una base de datos PostgreSQL. Se utilizó el marco de evaluación propuesto en el trabajo de tesis **"Integridad en DBMS Relacionales. Casos de Estudio: Access, MySQL, SQL Server."** [5] y se obtuvieron los siguientes resultados:

REGLA DE INTEGRIDAD DE ENTIDADES			
RESTRICCIÓN DE INTEGRIDAD	**ACCIÓN REFERENCIAL / OPCIÓN**	**CUMPLE / ACEPTA / VERIFICA**	**OBSERVACIONES Y VERIFICACIONES**
Definición de llaves primarias		**Si**	Se debe seleccionar un campo simple como llave primaria.
Inserción de clave primaria de un solo componente	No nulos	**Si**	No permite valores nulos en un campo de llave primaria.
	Valores repetidos	**Si**	Los valores deben ser únicos en la columna.
Inserción de clave primaria de más de un componente.	No nulos	**Si**	No permite valores nulos en alguno de los campos que componen la llave primaria
	Valores repetidos	**Si**	No permite valores repetidos en el conjunto que forman la llave primaria.
Actualización de clave primaria de un solo componente.	No nulos	**Si**	Cuando se actualiza el valor de la llave primaria, el valor no debe ser nulo.
	Valores repetidos	**Si**	Cuando se actualiza el valor de la llave primaria, el valor no debe existir en la columna.
Actualización de clave primaria de más de un componente.	No nulos	**Si**	Cuando se actualiza el valor de la llave primaria, no se permiten valores nulos en alguno de los campos que componen la llave primaria
	Valores repetidos	**Si**	Cuando se actualiza el valor de la llave primaria, no se permiten valores repetidos en el conjunto que forman la llave primaria.

Tabla 5.1 Pruebas de Integridad de entidades

REGLA DE INTEGRIDAD REFERENCIAL			
RESTRICCIÓN DE INTEGRIDAD	**ACCIÓN REFERENCIAL / OPCIÓN**	**CUMPLE / ACEPTA / VERIFICA**	**OBSERVACIONES Y VERIFICACIONES**
Definición de llaves foráneas		**Si**	Se define la llave foránea mediante un nombre específico, el cual, no debe existir.
Soporte de Integridad Referencial		**Si**	Se deben crear tuplas en donde el campo de llave primaria exista en la relación referida
Eliminación (DELETE)	Restrict	**Si**	Se requiere integridad referencial.
	Cascade	**Si**	Se debe restringir la eliminación en cascada.
	Set Null	**Si**	Siempre que no exista restricción que lo impida.
Actualización (UPDATE)	Restrict	**Si**	Se requiere integridad referencial.
	Cascade	**Si**	Se debe restringir la actualización en cascada.
	Set Null	**Si**	Siempre que no exista restricción que lo impida.

Tabla 5.2 Pruebas de Integridad Referencial

RESTRICCIONES DE RECHAZO			
RESTRICCIÓN DE INTEGRIDAD	**ACCIÓN REFERENCIAL / OPCIÓN**	**CUMPLE / ACEPTA / VERIFICA**	**OBSERVACIONES Y VERIFICACIONES**
Restricción CHECK		**Si**	Se debe de insertar una regla de validación en el diseño de una relación.

Tabla 5.3 Pruebas de Restricciones de Rechazo

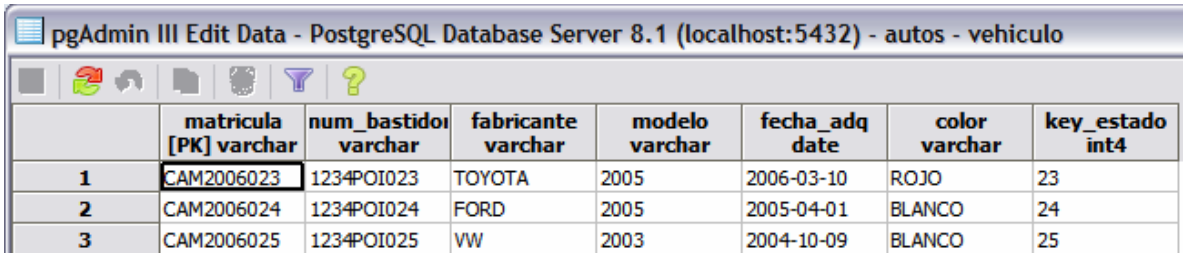
OTRAS RESTRICCIONES IMPORTANTES			
RESTRICCIÓN DE INTEGRIDAD	**ACCIÓN REFERENCIAL / OPCIÓN**	**CUMPLE / ACEPTA / VERIFICA**	**OBSERVACIONES Y VERIFICACIONES**
Restricción UNIQUE		**Si**	Se debe de especificar el atributo como unique.
Definición DEFAULT		**Si**	Se debe especificar el valor default en el diseño de la tabla.
Restricción de Tipo		**Si**	Se pueden crear tipos definidos por el usuario (dominio) para hacer referencia a algún tipo.
Disparador		**Si**	Se debe crear primero una función trigger y posteriormente el disparador.
Reglas definidas por el Usuario		**Si**	Se puede implementar algunos tipos de restricciones (constraints).

Tabla 5.4 Pruebas de otras restricciones

5.2 Análisis de los resultados de la evaluación de integridad

Definición de llaves primarias y llaves foráneas

Funcionamiento: OK.



	matricula [PK] varchar	num_bastidor varchar	fabricante varchar	modelo varchar	fecha_adq date	color varchar	key_estado int4
1	CAM2006023	1234POI023	TOYOTA	2005	2006-03-10	ROJO	23
2	CAM2006024	1234POI024	FORD	2005	2005-04-01	BLANCO	24
3	CAM2006025	1234POI025	VW	2003	2004-10-09	BLANCO	25

Figura. 5.5 Tabla Vehículo.

PostgreSQL permite la declaración de llaves primarias y foráneas, al desplegar la tabla, señala con las letras PK la columna que contiene a las llaves primarias. En la siguiente figura se muestra la tabla vehículo, que contiene llaves primarias y foráneas.

```
-- Table: vehiculo

-- DROP TABLE vehiculo;

CREATE TABLE vehiculo
(
  matricula varchar(10) NOT NULL,
  num_bastidor varchar(50) NOT NULL,
  fabricante varchar(20) NOT NULL,
  modelo varchar(20) NOT NULL,
  fecha_adq date NOT NULL,
  color varchar(10) NOT NULL,
  key_estado int4 NOT NULL,
  CONSTRAINT "Matricula" PRIMARY KEY (matricula),
  CONSTRAINT key_estado FOREIGN KEY (key_estado)
      REFERENCES estado (key_estados) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT bastidor UNIQUE (num_bastidor)
)
WITHOUT OIDS;
```

Tabla 5.6 Llaves primarias y foráneas.

Soporte de regla de Integridad de Entidades
Funcionamiento: OK.

PostgreSQL cuenta con soporte de Integridad de Entidades, ya que no permite la inserción de un campo de llave primaria como nulo. Al realizar la verificación el manejador le agrega la restricción de no aceptar nulos en los campos indicados. Además, de que cuando se crea una nueva tabla y no se le asigna algún campo como llave primaria (PK), el manejador envía un mensaje de advertencia, sugiriendo establecerla, mostrando las ventajas de contar con ella

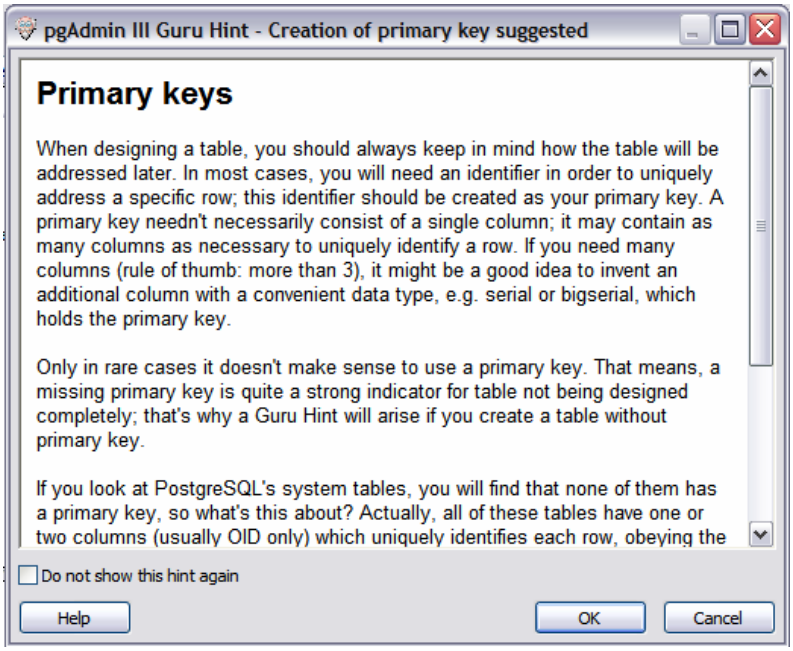
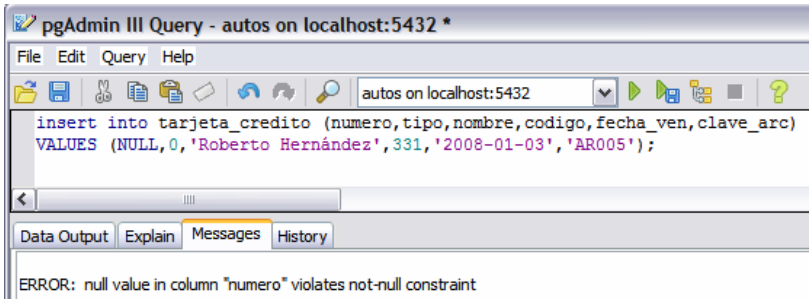


Figura. 5.7 Sugerencia de creación de llave primaria.

Inserción de llave primaria de un componente (NO NULOS)
Funcionamiento: OK.

PostgreSQL verifica que el valor de la llave primaria de un componente no sea nulo, esto se justifica al intentar insertar un valor nulo, como en el siguiente ejemplo:



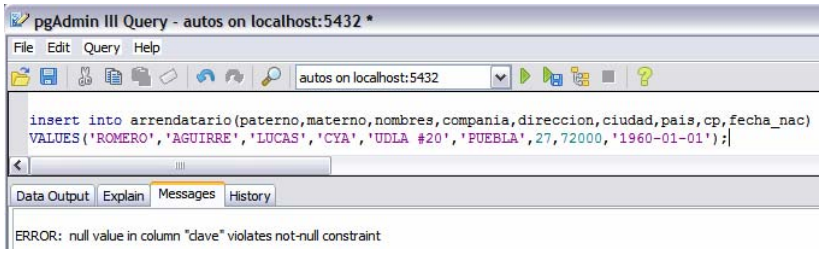


Figura. 5.8 Intento de insertar un valor nulo en un campo de llave primaria.

El manejador no permite el ingreso de valores nulos, pero al insertar cadenas vacías en un campo de llave primaria las representa como comillas ‘’. Al intentar reingresar otra cadena vacía como llave primaria, PostgreSQL no lo permite por violar la duplicidad de valores.

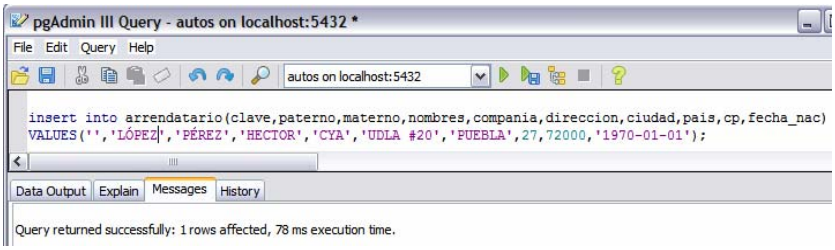


Figura. 5.9 Intento de insertar una cadena vacía en un campo de llave primaria.

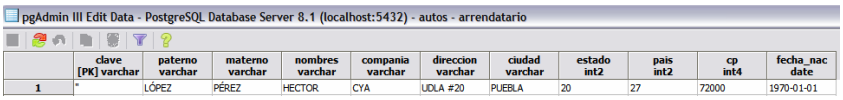


Figura. 5.10 Cadena vacía en un campo de llave primaria.

Inserción de llave primaria de un componente (VALORES REPETIDOS)

Funcionamiento: OK.

PostgreSQL verifica que no se ingrese un valor en el campo de llave primaria que ya existe. Si existiera algún intento, se envía un mensaje de error, informando que se intenta ingresar un valor repetido.

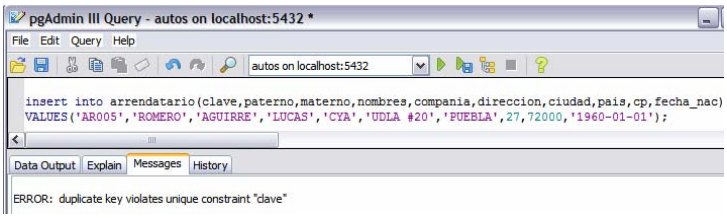


Figura. 5.11 Intento de insertar un valor repetido en un campo de llave primaria.

Inserción de llave primaria de más de un componente (NO NULOS)
Funcionamiento: OK.

PostgreSQL verifica que el valor de la llave primaria de más de un componente no sea nulo, es decir, que si al menos un componente de la llave primaria tiene un valor nulo, el manejador evitará el ingreso de los valores, esto se justifica al intentar insertar un valor nulo, como en el siguiente ejemplo:

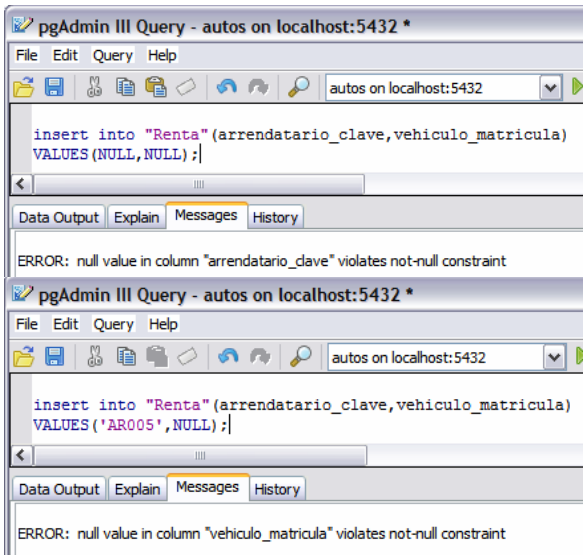


Figura. 5.12 Intento de insertar un valor nulo en un campo de llave primaria de más de un componente.

Inserción de llave primaria de más de un componente (VALORES REPETIDOS)
Funcionamiento: OK.

PostgreSQL verifica que no se ingrese un conjunto de valores que conforman la llave primaria que ya existen. Si existiera algún intento, se envía un mensaje de error, informando que se intenta ingresar un conjunto de valores repetidos en la llave primaria.

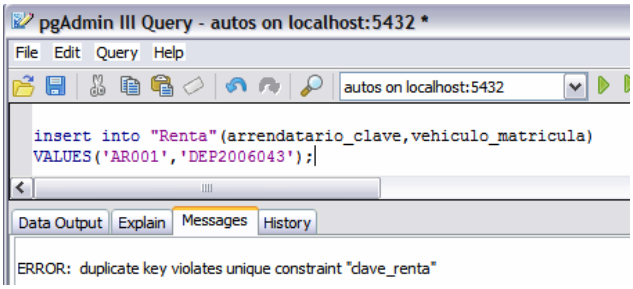


Figura. 5.13 Intento de insertar valores repetidos en un campo de llave primaria.

Actualización de llave primaria de un componente (NO NULOS)
Funcionamiento: OK.

Al intentar actualizar un valor de llave primaria por un valor nulo, el manejador envía un mensaje indicando que existe un error con la restricción **"not null"**.

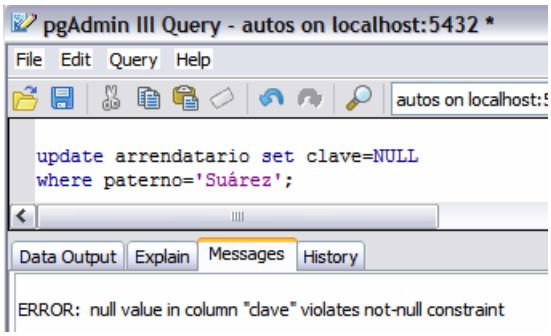


Figura 5.14 Intento de actualizar un campo de llave primaria por un valor nulo.

Actualización de llave primaria de un componente (VALORES REPETIDOS)
Funcionamiento: OK.

Al intentar actualizar un valor de llave primaria por un valor que ya existe en la tabla, el manejador envía un mensaje indicando que el valor esta repetido.

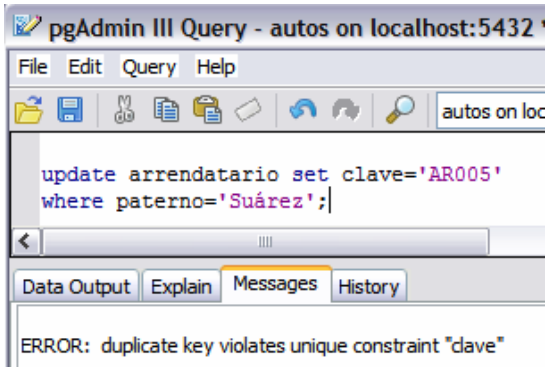


Figura 5.15 Intento de actualizar un campo de llave primaria por un valor repetido.

Actualización de llave primaria de más de un componente (NO NULOS)

Funcionamiento: OK.

Al intentar actualizar un valor de llave primaria por un valor nulo, el manejador envía un mensaje indicando que existe un error con la restricción "**not null"**.

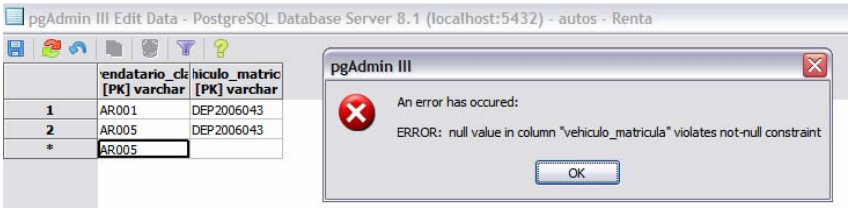


Figura 5.16 Intento de actualizar un campo de llave primaria de más de un componente por un valor nulo.

Actualización de llave primaria de más de un componente (VALORES REPETIDOS)

Funcionamiento: OK.

Al intentar actualizar un valor de llave primaria de más de un componente por un valor que ya existe en la tabla, el manejador envía un mensaje indicando que el valor esta repetido.

Figura 5.17 Intento de actualizar un campo de llave primaria de más de un componente por una combinación de valores repetidos.

Soporte de integridad referencial
Funcionamiento: OK.

El manejador verifica que la regla de integridad referencial se cumpla. PostgreSQL comprueba que el valor de la llave foránea a la que se hace referencia exista en la relación. Si este valor no existe, impide la transacción, y envía un mensaje de error. Veamos un ejemplo: Si deseamos ingresar una nueva tupla en la tabla Email, pero la clave del arrendatario no existe, nos enviará un mensaje de error.

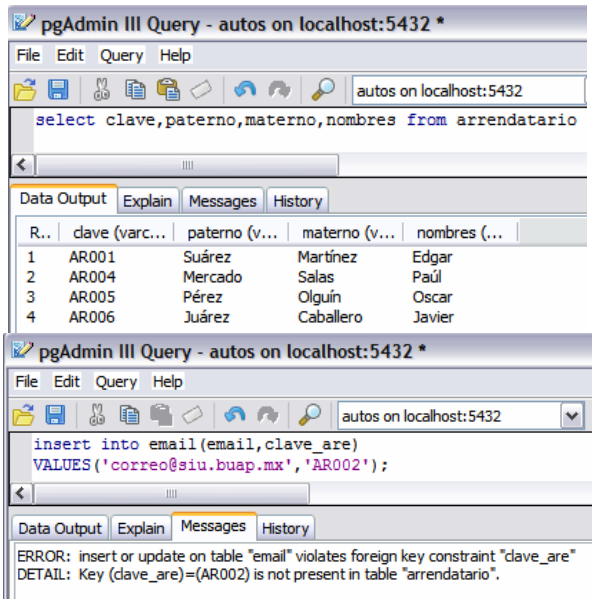
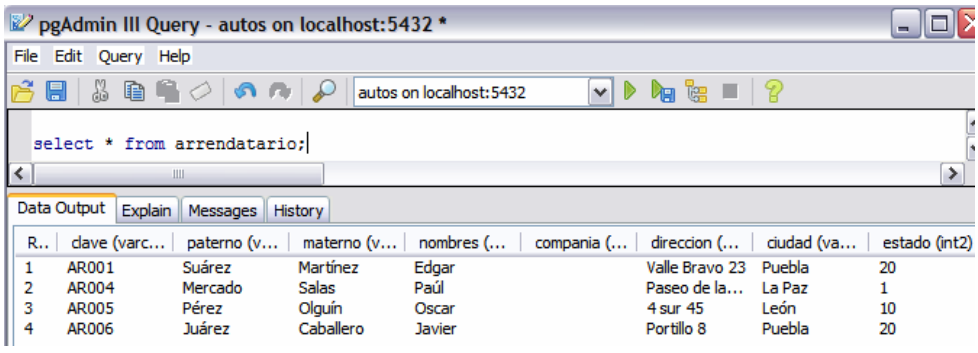


Figura 5.18 Verificación de integridad referencial.

Eliminación (RESTRICT, CASCADE, NULL)
Funcionamiento: OK.

PostgreSQL realiza la eliminación de tuplas en forma restringida, en cascada y puesta a nulos de forma correcta.

- **Eliminación restringida.**



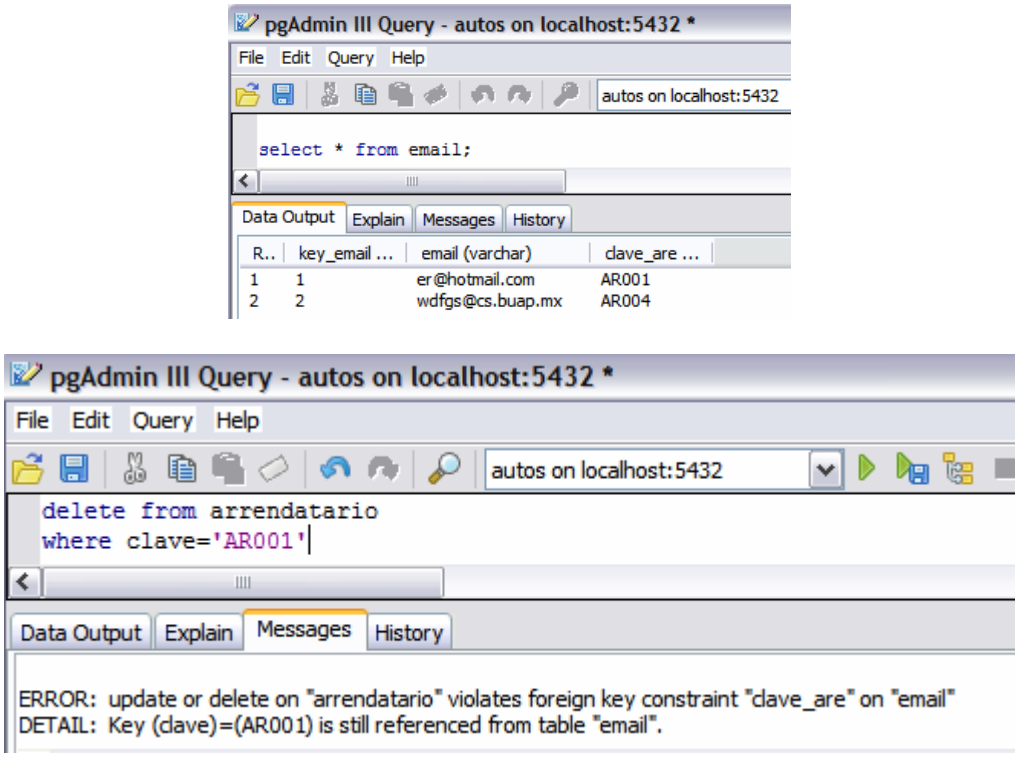
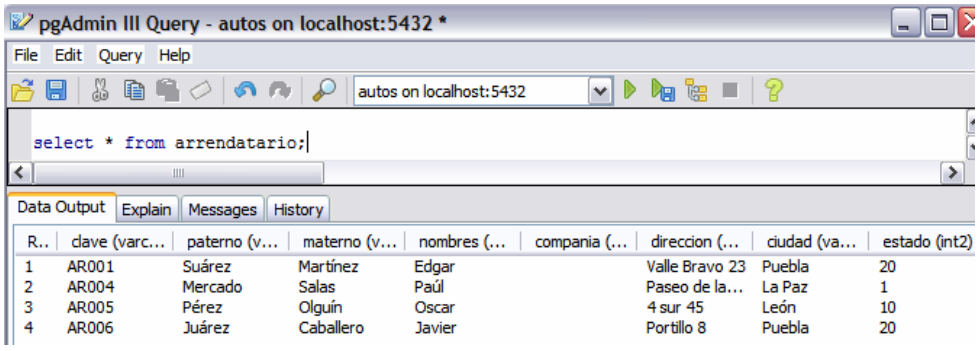


Figura 5.19 Eliminación de tuplas restringida.

- **Eliminación en cascada.**



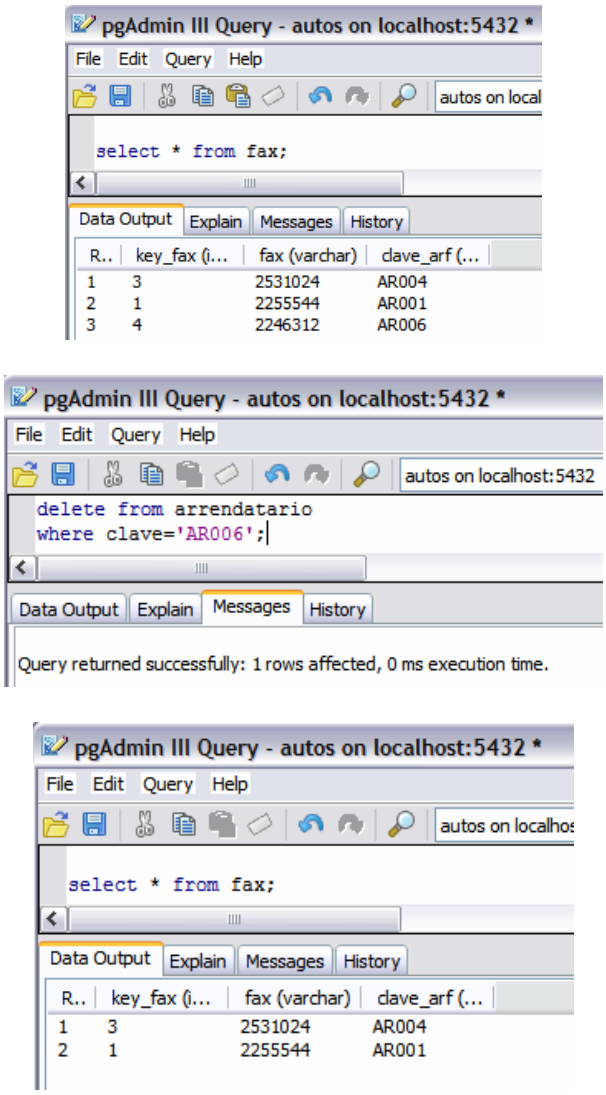
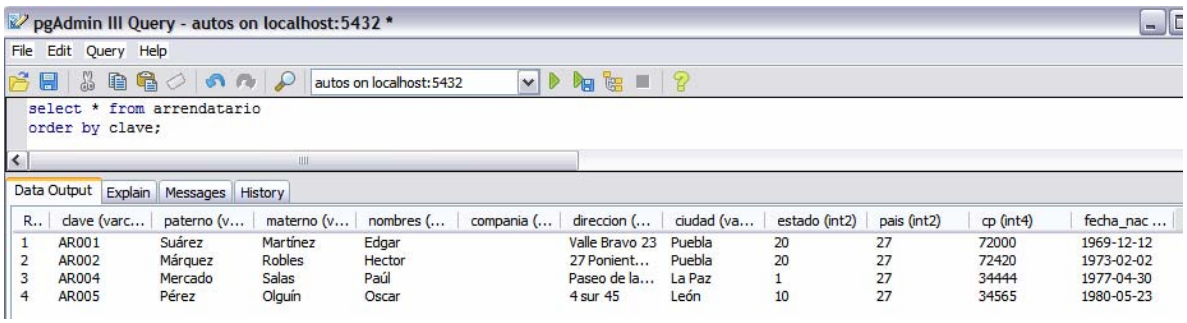


Figura 5.20 Eliminación de tuplas en cascada.

- **Eliminación con puesta a nulos.**



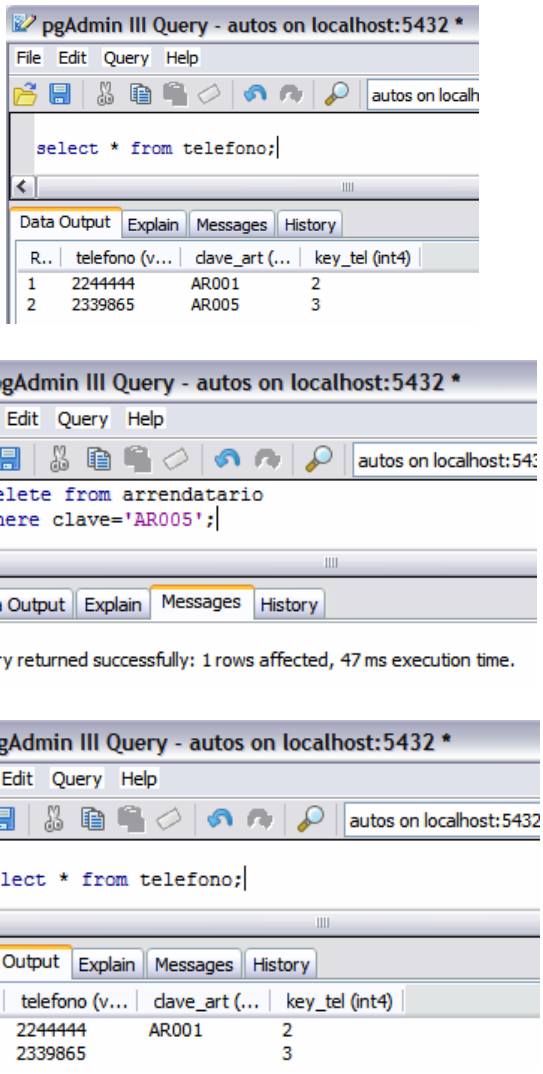


Figura 5.21 Eliminación de tuplas con valores puestos a nulos.

Actualización (RESTRICT, CASCADE, NULL)

Funcionamiento: OK.

PostgreSQL realiza la actualización de tuplas en forma restringida, en cascada y puesta a nulos de forma correcta.

- **Actualización restringida.**

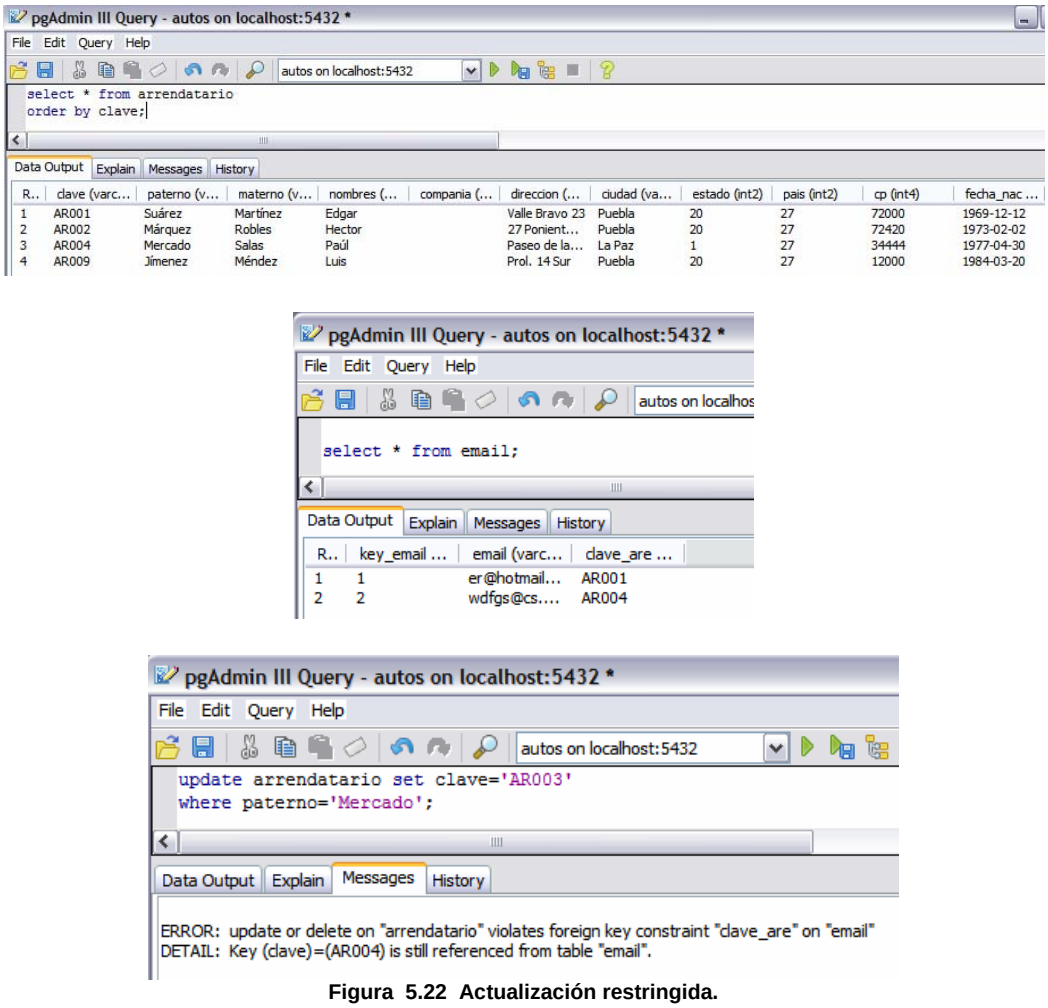
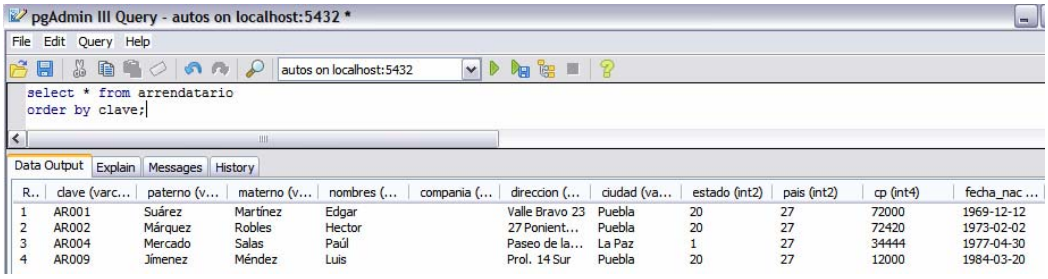


Figura 5.22 Actualización restringida.

- **Actualización en cascada.**



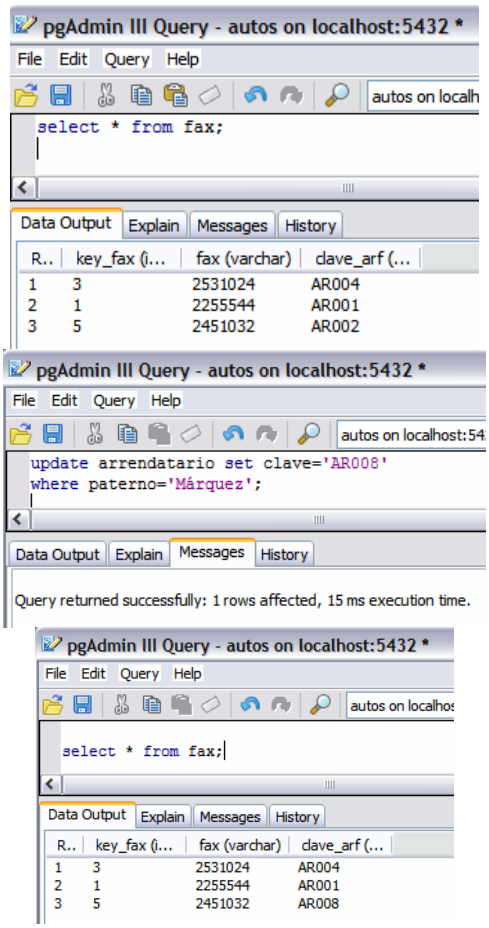
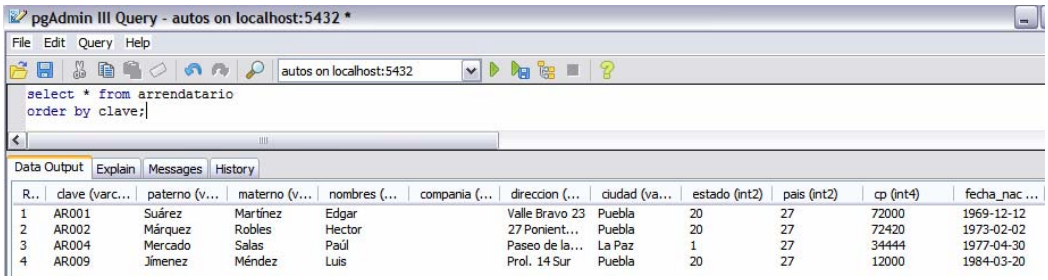


Figura 5.23 Actualización en cascada.

- **Actualización con puesta a nulos.**



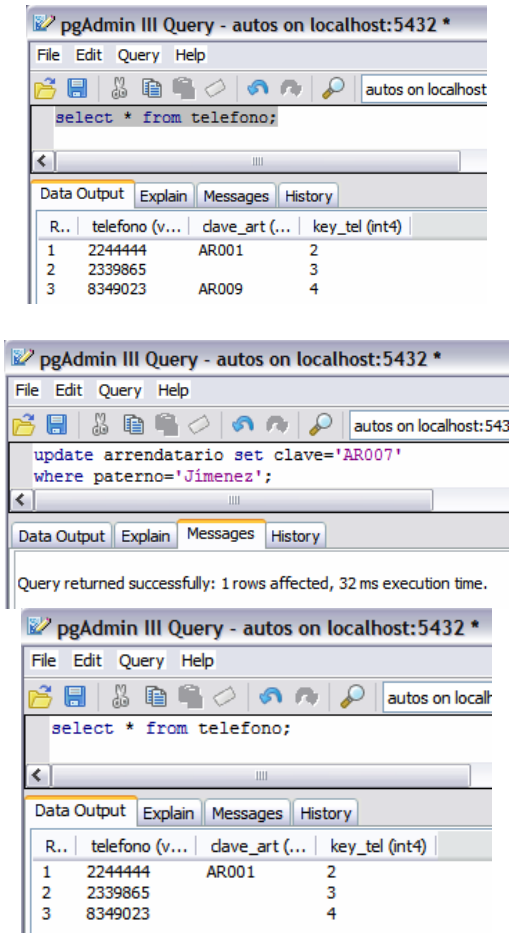


Figura 5.24 Actualización con valores puestos a nulos.

Restricción CHECK.

Funcionamiento: OK.

PostgreSQL verifica correctamente las restricciones **"CHECK"**, en este caso, el arrendatario para poder rentar un auto debe ser mayor de 18 años, y se específica como se muestra a continuación:

```
CREATE TABLE arrendatario
(
  clave varchar(5) NOT NULL,
  paterno varchar(30) NOT NULL,
  materno varchar(30),
  nombres varchar(50) NOT NULL,
  compania varchar(70),
  direccion varchar(40) NOT NULL,
  ciudad varchar(30) NOT NULL,
  estado int2 NOT NULL DEFAULT 20,
  pais int2 NOT NULL,
  cp int4,
  fecha_nac date NOT NULL,
  CONSTRAINT clave PRIMARY KEY (clave),
  CONSTRAINT date_nac CHECK (fecha_nac::text < (1990 - 1 - 1)::text)
)
WITHOUT OIDS;
```

Tabla 5.25 Tabla arrendatario con restricción Check.

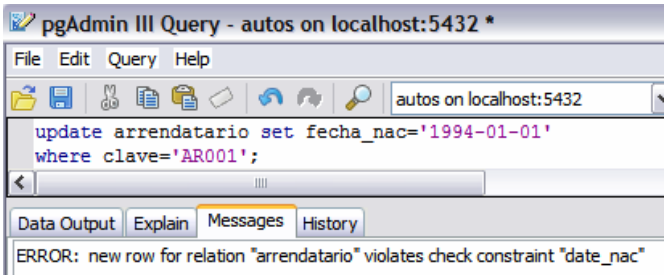


Figura 5.26 Restricción CHECK.

Valores únicos (UNIQUE)

Funcionamiento: OK.

El manejador verifica perfectamente la restricción "**Unique",** comprobando que los valores no se repitan en la columna.

```
CREATE TABLE vehiculo
(
  matricula varchar(10) NOT NULL,
  num_bastidor varchar(50) NOT NULL,
  fabricante varchar(20) NOT NULL,
  modelo varchar(20) NOT NULL,
  fecha_adq date NOT NULL,
  color varchar(10) NOT NULL,
  key_estado int4 NOT NULL,
  CONSTRAINT "Matricula" PRIMARY KEY (matricula),
  CONSTRAINT key_estado FOREIGN KEY (key_estado)
     REFERENCES estado (key_estados) MATCH SIMPLE
     ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT bastidor UNIQUE (num_bastidor)
)
WITHOUT OIDS;
```

Tabla 5.27 Valores únicos.

Valores por default (DEFAULT)

Funcionamiento: OK.

El manejador da el valor por "**default"** a un campo, si a este no se le asigna un valor específico. En el siguiente ejemplo se le asigna el valor 20 al estado, que corresponde a Puebla.

```
CREATE TABLE arrendatario
(
  clave varchar(5) NOT NULL,
  paterno varchar(30) NOT NULL,
  materno varchar(30),
  nombres varchar(50) NOT NULL,
  compania varchar(70),
  direccion varchar(40) NOT NULL,
  ciudad varchar(30) NOT NULL,
  estado int2 NOT NULL DEFAULT 20,
  pais int2 NOT NULL,
  cp int4,
  fecha_nac date NOT NULL,
  CONSTRAINT clave PRIMARY KEY (clave),
  CONSTRAINT date_nac CHECK (fecha_nac::text < (1990 - 1 - 1)::text)
)
WITHOUT OIDS;
```

Tabla 5.28 Valores por default.

Restricciones de tipo (Dominio)

Funcionamiento: OK.

PostgreSQL permite definir tipos de datos definidos por el usuario, esto es, definir el dominio de los datos y creas tipos de datos personalizados.

```
CREATE DOMAIN "Dominio"
  AS bytea;

CREATE TABLE documento
(
  tarjeta bytea,
  comprobante bytea NOT NULL,
  identificacion bytea NOT NULL,
  clave_ard varchar(5),
  key_doc int4 NOT NULL,
  licencia "Dominio" NOT NULL,
  CONSTRAINT key_doc PRIMARY KEY (key_doc),
  CONSTRAINT clave_ard FOREIGN KEY (clave_ard)
     REFERENCES arrendatario (clave) MATCH SIMPLE
     ON UPDATE CASCADE ON DELETE CASCADE
)
WITHOUT OIDS;
```

Tabla 5.29 Tipos de datos definidos por el usuario.

Disparadores (Triggers)

Funcionamiento: OK.

Los disparadores son funciones que se ejecutan de forma automática en respuesta a ciertos eventos que ocurren en la base de datos.

En PostgreSQL, se pueden ejecutar:

- Antes o después de una inserción (INSERT).
- Antes o después de una actualización (UPDATE).
- Antes o después de un borrado (DELETE).

La definición de un disparador consta de dos partes:

- Una función asociada a un disparador.
- La definición del disparador, esto es, de que tabla se esperan los eventos y a que tipo de evento se responderá.

PostgreSQL, permite la definición de disparadores, al crear por primera vez una función trigger, hay que cargar el lenguaje PLPGSQL.

La función asociada al disparador es:

```
CREATE OR REPLACE FUNCTION renta_tri()
  RETURNS "trigger" AS
$BODY$
BEGIN
if (TG_OP='UPDATE') THEN
INSERT INTO catalogo_renta VALUES(
OLD.arrendatario_clave,OLD.vehiculo_matricula,current_date);
END IF;
RETURN NULL;
END;
$BODY$
  LANGUAGE 'plpgsql' VOLATILE;
```

Tabla 5.30 Función trigger.

La definición del disparador es la siguiente, y se encarga de utilizar la función trigger y al realizar una actualización en la tabla “Renta”, copia los valores anteriores a la tabla catalogo_renta.

```
CREATE TRIGGER actualizar_renta
  BEFORE UPDATE
  ON "Renta"
  FOR EACH ROW
  EXECUTE PROCEDURE renta_tri();
COMMENT ON TRIGGER actualizar_renta ON "Renta" IS 'antes de actualizar guarda en el
catalogo';
```

Tabla. 5.31 Disparador.

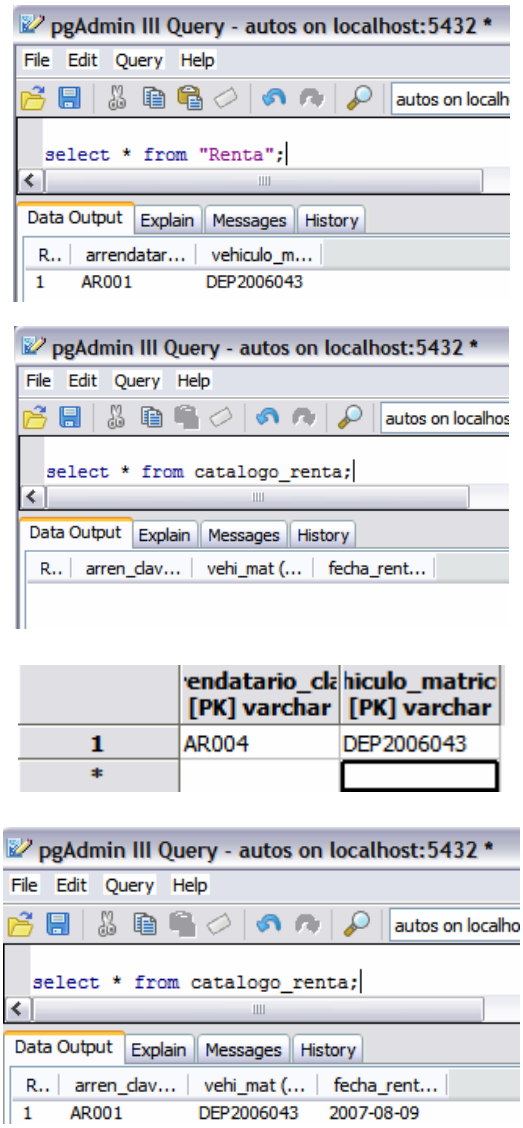


Figura 5.32 Prueba con disparadores.

Reglas definidas por el usuario

Funcionamiento: OK.

Son muchas las cosas que se hacen utilizando disparadores que pueden hacerse también utilizando el sistema de las reglas de PostgreSQL. Si se necesitan comprobaciones para valores válidos, y en el caso de aparecer un valor inválido dar un mensaje de error, eso deberá hacerse por ahora con un trigger. Para los tratamientos que podrían implementarse de ambas formas, dependerá del uso de la base de datos cuál sea la mejor. Un trigger se dispara para cada fila afectada. Una regla manipula el árbol de traducción o genera uno adicional.

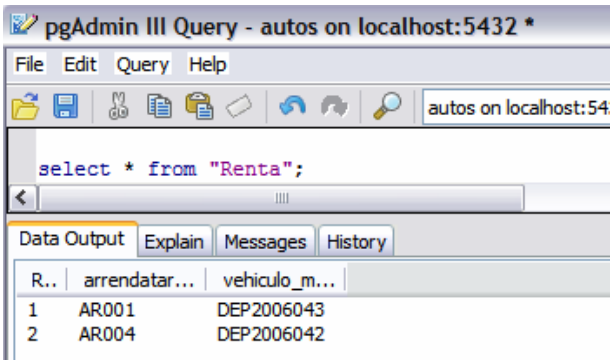
En el siguiente ejemplo se muestra el uso de las reglas de PostgreSQL, utilizaremos las tablas anteriores. Al borrar una tupla en la tabla Renta, la regla borrará el registro correspondiente en la tabla catálogo_renta.

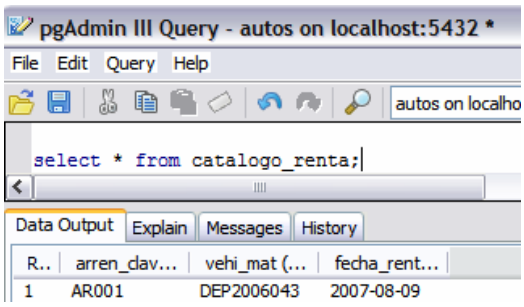
CREATE OR REPLACE RULE regla_renta **AS**

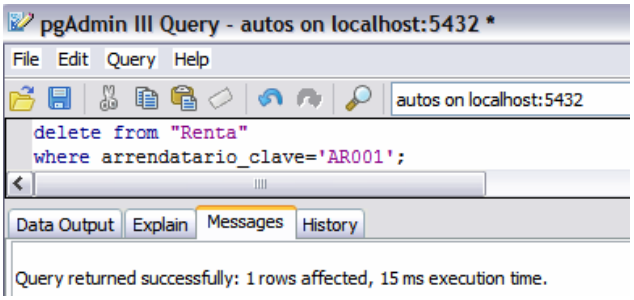
ON DELETE TO "Renta" **DO DELETE FROM** catalogo_renta

WHERE catalogo_renta.arren_clave::text = old.arrendatario_clave::text;

Tabla 5.33 Reglas de PostgreSQL.







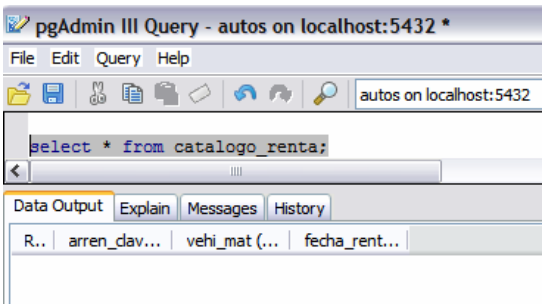


Figura 5.34 Ejecución de la regla de PostgreSQL.

CONCLUSIONES

PostgreSQL lo puedo catalogar como un manejador objeto relacional, ya que un manejador orientado a objetos, almacena objetos abstractos, en cambio, en PostgreSQL se siguen almacenando tuplas. Además PostgreSQL, soporta Polimorfismo Procedural y permite que una función o procedimiento almacenado pueda ser utilizado en cualquier parte de una consulta.

PostgreSQL antes llamado Postgres95, ha venido mejorándose a través del tiempo, con aportaciones de usuarios de todo el mundo, gracias a que cuenta con una licencia BSD, que permite modificar el código y redistribuirlo, por lo cual se tienen versiones frecuentemente. Esto es una causa por la que muchas empresas están migrando sus sistemas a este manejador, dejando a un lado los productos de Microsoft y Oracle. Además, de tener un gran desempeño con bases de datos de gran tamaño, siendo una alternativa para usuarios de MySql, que necesiten un manejador de más capacidad.

El presente trabajo documenta las opciones que proporciona este manejador de base de datos objeto relacional con respecto al soporte de integridad, para ofrecer un apoyo al incluir el aspecto de integridad en el diseño de las bases de datos, y reducir esta verificación en el código de las aplicaciones, con el fin de reducir costos de programación y asegurar la calidad de los datos.

Después de haber realizado el análisis de integridad y evaluar el manejador de base de datos pude concluir que PostgreSQL cumple con el soporte de integridad propuesto en el marco de evaluación al que fue sometido. Regularmente los manejadores de base de datos orientados a objetos realizan la verificación de la integridad de los datos mediante código procedural que agregan los desarrolladores, pero este manejador incluye la mayoría de las propiedades y restricciones de integridad de un manejador relacional.

En los cuatro grupos en los que se encuentran divididas las reglas de integridad (reglas de integridad de entidades, reglas de integridad referencial, restricciones de rechazo y otras restricciones de integridad importantes), pude darme cuenta que el DBMS realiza una aseguramiento de la información, enviando mensajes de error o advertencias, cuando se viola alguna de estas restricciones. Además el manejador puede crear reglas de integridad particulares para casos específicos, tipos de datos definidos por el usuario, utilizar tipos **blob** (objetos muy grandes) y utilizando su modelo MVCC (Multiversión) maneja internamente integridad transaccional, es decir, que mientras se consulta una Base de Datos, cada transacción ve una imagen de los datos (una versión de la Base de Datos), sin tener en cuenta el estado actual de los datos. Esto evita que la transacción vea datos inconsistentes que pueden ser causados por la actualización de otra transacción concurrente en la misma fila de datos, proporcionando aislamiento transaccional para cada sesión de la base de datos.

Al ser un manejador objeto relacional, utilice la propiedad de la herencia en algunas tablas y pude darme cuenta que tiene un buen desempeño. Otra ventaja es que puede convivir con distintos sistemas operativos (Unix, Windows, Novell, OS2) y con varios lenguajes de programación (PHP, Perl, TCL, Python,C++,Visual Basic).

BIBLIOGRAFÍA

[1] Codd. E.F. “A Relational Model of Data for Large Shared Data Banks”. CAMC 13, núm.6 (junio 1970). Publicado otra vez Milestones of Reseca –Selected Papers 1958-1982 (CAMC, ejemplar del 25 aniversario), CACM 26, núm. 1 (enero de 1983).

[2] Date C.J. “Introducción a los Sistemas de Bases de Datos”. Vol. 1, 7ª. Ed. 2001, Pearson-Educación de México, S.A. de C.V.

[3] Date C.J. “Introducción a los Sistemas de Bases de Datos”. Vol. 1, 5ª. Ed. 1998, Addison Wesley Longman de México S.A. de C.V.

[4] Ullman J.D. & J. Widom, “Introducción a los Sistemas de Bases de Datos”. Prentice-Hall, México, 1999.

[5] Galindo-Aburto Ercilia. ,“Integridad en DBMS Relacionales. Casos de Estudio: Access, MySQL, SQL Server.”. Asesor: Ma. del Rocío Boone Rojas. Tesis de Licenciatura, Julio-2004, BUAP, Fac. de Cs. de la Computación, Pue. México.

[6] Boone-Rojas Ma. del Rocío. “Especificaciones, Extensiones y Ramificaciones del Modelo Relacional”, Proyecto de Investigación Interno. 2004, BUAP, Fac. de Cs. de la Computación, Secretaría de Investigación y Estudios de Posgrado. Pue. México.

[7] Bernabé-Loranca Beatriz. ,“Procesador para Álgebra Relacional”. Asesor: Ma. del Rocío Boone Rojas. Tesis de Licenciatura 1994, BUAP, Fac. de Cs. de la Computación, Pue. México.

[8] Bernabé-Loranca, M.B. & Boone-Rojas, M. R.; “Especificaciones Complementarias para el Soporte de Integridad en DBMS relacionales”, CIC 2004, Congreso Internacional de Computación, IPN, México, DF, Octubre 2004.

[9] Boone-Rojas Rocío, Soriano-Ulloa M.A., Bernabé-Loranca, “Marco de Evaluación para el Componente de Integridad en DBMS relacionales”. II Congreso Nal. de Cs. de la Computación. Puebla, Pue. Méx. Nov. 2004.

[10] Pastor-López O, Pedro Blesa P.; “Gestión de Bases de Datos”, Universidad Politécnica de Valencia. Servicios de Publicaciones, Valencia 2000.

[11] Pastor-López O et. al. “Análisis de Restricciones de Integridad en el Nivel Conceptual”, VII Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes Software (IDEAS 2004), Arequipa, Perú mayo 2004

[12] Boone-Rojas, & M. R. Bernabé-Loranca, M.B.; “Especificaciones y Gramática para el Soporte de Integridad en RDBMS”, Congreso COINELECOMP Febrero 2005 UDLA, Puebla, Pue. Méx..

[13] Silberschatz A. & Korth, H.F. “Fundamentos de Bases de Datos”, 4ª Edición, Mc. Graw Hill.

[14]Bernabé-Loranca, Boone-Rojas Rocío, López-Sales Ramiro, “Extensiones Gramaticales del Álgebra Relacional para el Soporte de Integridad en Sistemas Relacionales”, 6º. Congreso Estudiantil de Computación, CORE 05, IPN Mex. D.F. 12, 13 Mayo /05.

[15] Montero-Ramírez Isaac, Boone-Rojas Rocío, Bernabé-Loranca, Carrillo-Ruiz Maya, Soriano-Ulloa M.A., “Marco de Evaluación para el Subsistema de Seguridad de DBMS Relacionales”, *VI Encuentro Internacional de Computación, ENC 05Carteles,* SMCC (Mexican Society of Computer Science), 26 – 30 Septiembre 05, Pue. Pue. Mex.

[16] Sistemas de Base de datos. http://www.acm.org/crossroads/espanol/xrds7-3/ordbms.html

[17] Programación Orientada a Objetos, Oracle y Sql Server. http://www.solomanuales.org/curso-MatriculaShow.cfm?id_curso=5106404003225050515668545667455 5&id_centro=4320411002146656557067695052455 0&Nombre=Javier%20Caballero&Mail=racim_f@hotmail.com&Pais=7&Provincia=360

[18] Equipo de Desarrollo de PostgreSQL, Thomas Lockhart, “Tutorial de PostgreSQL”.

[19] Equipo de Desarrollo de PostgreSQL, Thomas Lockhart, “Guía del Usuario de PostgreSQL”.

[20] Equipo de Desarrollo de PostgreSQL, Thomas Lockhart, “Guía del Administrador de PostgreSQL”.

[21] Michael Stonebraker, Dorothy Moore, “Object-relational DBMSs : the next great wave”, Ed. San Francisco, Calif. : Morgan Kaufmann Publishers, ©1996.

[22] PostgreSQL. http://www.postgresql.org.mx/?q=node/8

[23] Documentación de PostgreSQL. http://es.tldp.org/Postgresql-es/web/

[24] El Sistema de reglas de Postgres. http://es.tldp.org/Postgresql-es/web/navegable/programmer/x1201.html

GLOSARIO

A

Atributo: *Es una propiedad o característica de un elemento; es una columna de una tabla.*

B

Base de Datos: *Es un conjunto organizado de datos que guardan cierta relación entre si, permitiendo el fácil y rápido acceso a la información.*

… Deductivas: *Son aquellas bases de datos que permiten hacer deducciones a través de inferencias. Se basan principalmente en Reglas y Hechos que son almacenados en la Base de Datos.*

… en Red: *Se basan en el concepto de nodo, permitiendo que el mismo nodo tenga varios padres, evitando el concepto de redundancia de datos.*

… Jerárquicas: *Almacenan la información en forma jerárquica. Los datos se organizan en una forma similar a un árbol, en donde un nodo padre de la información puede tener varios hijos. El nodo que no tiene padre es llamado raíz, y a los nodos que no tienen hijos son llamados hojas.*

… Orientadas a Objetos: *Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos:*

- ***Encapsulamiento:*** *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.*

… Relacionales: *Su idea fundamental es el uso de relaciones. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados tuplas .Pensando en cada relación como una tabla que esta compuesta por registros (filas) que representan las tuplas y campos (columnas) que representan sus atributos.*

C

Clase: *Conjunto de objetos que comparten los mismos atributos, métodos y funcionamiento.*

Clave candidata: *Son aquellos atributos que son candidatos a ser claves primarias.*

Clave foránea: *Es aquella columna de una tabla que hace referencia a*

una clave primaria de otra tabla.

Clave Primaria: *Es un campo o un conjunto de campos, que identifica de manera única a cada registro.*

D

Datos Persistentes: *Son los datos que siguen existiendo una vez que el programa que los creo ha concluido*

DBA (Administrador de la Base de Datos): *Persona o grupo de personas responsable de la definición, protección y eficiencia de la base de datos de una empresa, al ser colocada en una computadora.*

DBMS (Sistema Manejador de Bases de Datos): *Son un tipo de software muy específico, dedicado a servir de interfaz entre las bases de datos y las aplicaciones que la utilizan*

I

Integridad: *Se refiere a la seguridad de que la información no ha sido alterada.*

… Referencial: *Garantiza que una entidad (fila o registro) siempre se relaciona con otras entidades válidas, es decir, que existen en la base de datos.*

L

Lenguajes de Cuarta Generación: *El programador no incorpora el procedimiento a seguir, ya que el propio lenguaje es capaz de indicar al ordenador cómo debe ejecutar el programa, Los lenguajes de cuarta generación son más fáciles de usar que los 3GL: suelen incluir interfaces gráficos y capacidades de gestión avanzadas, pero consumen muchos más recursos del ordenador que la generación de lenguajes previa.*

Los lenguajes SQL y QBE son ejemplos de 4GL. Hay otros tipos de 4GL: generador de formularios, informes, gráficos y de aplicaciones.

M

Método: *Es la implementación de un algoritmo que representa una operación o función que un objeto realiza. El conjunto de los métodos de un objeto determinan el comportamiento del objeto.*

Memoria swap: *Técnica que permite que una computadora simule más memoria principal de la que posee. La técnica es usada por la mayoría de los sistemas operativos actuales.*

Modelo Relacional:

El modelo relacional para la gestión de una base de datos es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos.

Éste es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos

dinámicamente. Tras ser postuladas su bases 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 base de datos.

En este modelo se representan los datos y las relaciones entre estos, a través de una colección de tablas, en las cuales los renglones (tuplas) equivalen a los cada uno de los registros que contendrá la base de datos y las columnas corresponden a las características(atributos) de cada registro localizado en la tupla.

O

Objeto: *Es una instancia de una clase, que encapsula estado y procesos. Pueden representar cosas reales o conceptuales.*

P

Programación Orientada a objetos: *Es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas*

R

Restricciones de Integridad: *Son las restricciones que se imponen para que la base de datos nunca llegue a un estado inconsistente*

S

Sistema de Gestión de Archivos: *Un sistema de gestión de archivos es aquel sistema software que provee servicios a los usuarios y aplicaciones en el uso de archivos. El único camino que tiene el usuario o la aplicación tiene para acceder a los archivos es a través de un sistema de gestión de archivos. Esto revela para el usuario o programador la necesidad de desarrollar software de propósito especial para cada aplicación y provee al sistema un medio de controlar su ventaja más importante.*

Estos son los objetivos de un sistema de gestión de archivos:

1. *Cumplir con las necesidades de gestión de datos y con los requisitos del usuario, que incluye el almacenamiento de, datos y la capacidad de ejecutar las operaciones en la lista precedente.*

2. *Garantizar, en la medida de lo posible, que el dato en el archivo es valido.*
3. *Optimizar el rendimiento, ambos desde el punto de vista del sistema en términos de productividad global, y como punto de vista del usuario en tiempos de respuesta.*
4. *Para proveer soporte de E/S para una variedad de tipos de dispositivos de almacenamiento.*
5. *Para minimizar o eliminar la posibilidad de perdida o destrucción de datos.*
6. *Para proveer un conjunto estándar de rutinas de E/S.*
7. *Para proveer soporte de E/S para múltiples usuarios, en caso de sistemas multiusuarios.*