**CAPITULO 2**

**MODELO RELACIONAL DE BASES DE DATOS Y LENGUAJE DE CONSULTAS**

**2.1 Introducción al modelo relacional**

Para realizar la introducción al modelo relacional se han considerado párrafos de los textos de Batini, Ceri y Navathe (1994) y de la dirección electrónica “<http://www.itlp.edu.mx>”.

El diseño de bases de datos es el proceso por el que se determina la organización de una base de datos, incluido su estructura, contenido y las aplicaciones que se han de desarrollar.

Durante mucho tiempo, el diseño de bases de datos fue considerado una tarea para expertos: más un arte que una ciencia. Sin embargo, se ha progresado mucho en el diseño de bases de datos y éste se considera ahora una disciplina estable, con métodos y técnicas propios. Debido a la creciente aceptación de las bases de datos por parte de la industria y el gobierno en el plano comercial, y a una variedad de aplicaciones científicas y técnicas, el diseño de bases de datos desempeña un papel central en el empleo de los recursos de información en la mayoría de las organizaciones.

El diseño de bases de datos ha pasado a constituir parte de la formación general de los informáticos, en el mismo nivel que la capacidad de construir algoritmos usando un lenguaje de programación convencional. Las últimas dos décadas se han caracterizado por un fuerte crecimiento en el número e importancia de las aplicaciones de bases de datos. Las bases de datos son componentes esenciales de los sistemas de información, usadas rutinariamente en todos los computadores. El diseño de bases de datos se ha convertido en una actividad popular, desarrollada no sólo por profesionales sino también por no especialistas. A finales de la década de 1960, cuando las bases de datos entraron por primera vez en el mercado del software, los diseñadores de bases de datos actuaban como artesanos, con herramientas muy primitivas: diagramas de bloques y estructuras de registros que eran los formatos comunes para las especificaciones. Además el diseño de bases de datos se confundía frecuentemente con la implantación de las bases de datos. Hoy en día esta situación ha cambiado: los métodos y modelos de diseño de bases de datos han evolucionado paralelamente con el progreso de la tecnología en los sistemas de bases de datos. Hemos incursionado en la era de los sistemas relacionales de bases de datos, que ofrecen poderosos lenguajes de consulta, herramientas para el desarrollo de aplicaciones e interfaces amables para los usuarios. La tecnología de bases de datos cuenta ya con un marco teórico, que incluye la teoría relacional de datos, procesamiento y optimización de consultas, control de concurrencia, gestión de transacciones y recuperación, etc. A través del tiempo la tecnología de bases de datos ha dado grandes pasos, es así que se han desarrollado metodologías y técnicas de diseño. Se ha alcanzado un consenso, por ejemplo, sobre la descomposición del proceso de diseño en fases, sobre los principales objetivos de cada fase y sobre las técnicas para conseguir estos objetivos.

Desafortunadamente, las metodologías de diseño de bases de datos no son muy populares; la mayoría de las organizaciones y de los diseñadores individuales confía muy poco en las metodologías para llevar a cabo el diseño y esto se considera, con frecuencia, una de las principales causas de fracaso en el desarrollo de los sistemas de información. Debido a la falta de enfoques estructurados para el diseño de bases de datos, a menudo se subestiman el tiempo o los recursos necesarios para un proyecto de bases de datos, las bases de datos son inadecuadas o ineficientes en relación a las demandas de la aplicación, la documentación es limitada y el mantenimiento es difícil. Muchos de estos problemas se deben a la falta de una claridad que permita entender la naturaleza exacta de los datos, a un nivel conceptual y abstracto. En muchos casos, los datos se describen desde el comienzo del proyecto en términos de las estructuras finales de almacenamiento; no se da peso a un entendimiento de las propiedades estructurales de los datos que sea independiente de los detalles de la realización.

**2.2 Estructura del modelo relacional**

El diseño de una base de datos es un proceso complejo que abarca decisiones a muy distintos niveles. La complejidad se controla mejor si se descompone el problema en subproblemas y se resuelve cada uno de estos subproblemas independientemente, utilizando técnicas específicas. Así, el diseño de una base de datos se descompone en diseño conceptual, diseño lógico y diseño físico. El diseño conceptual parte de las especificaciones de requisitos de usuario y su resultado es el esquema conceptual de la base de datos.

Un esquema conceptual es una descripción de alto nivel de la estructura de la base de datos, independientemente del sistema implementado para la base de datos, que se vaya a utilizar para manipularla. Un modelo conceptual es un lenguaje que se utiliza para describir esquemas conceptuales. El objetivo del diseño conceptual es describir el contenido de información de la base de datos y no las estructuras de almacenamiento que se necesitarán para manejar esta información.

El diseño lógico parte del esquema conceptual y da como resultado un esquema lógico. Un esquema lógico es una descripción de la estructura de la base de datos en términos de las estructuras de datos que puede procesar un tipo de sistema implementado para el manejo de la misma. Un modelo lógico es un lenguaje usado para especificar esquemas lógicos (modelo relacional, modelo de red, etc.). El diseño lógico depende del tipo de sistema implementado para el manejo de la base de datos que se vaya a utilizar, más no depende del producto concreto.

El diseño físico parte del esquema lógico y da como resultado un esquema físico. Un esquema físico es una descripción de la implementación de una base de datos en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos. Por ello, el diseño físico depende también del sistema implementado para el manejo de la base de datos concreto y el esquema físico se expresa mediante su lenguaje de definición de datos.

**2.2.1 Metodología del diseño conceptual**

El primer paso en el diseño de una base de datos es la producción del esquema conceptual. Normalmente, se construyen varios esquemas conceptuales, cada uno para representar las distintas visiones que los usuarios tienen de la información. Cada una de estas visiones suelen corresponder a las diferentes áreas funcionales de la empresa como, por ejemplo, producción, ventas, recursos humanos, etc. Estas visiones de la información, denominadas *vistas*, se pueden identificar de varias formas.

Una opción consiste en examinar los diagramas de flujo de datos, que se pueden haber producido previamente, para identificar cada una de las áreas funcionales. La otra opción consiste en entrevistar a los usuarios, examinar los procedimientos, los informes y los formularios, y también observar el funcionamiento de la empresa. A los esquemas conceptuales correspondientes a cada vista de usuario se les denomina *esquemas conceptuales locales*. Cada uno de estos esquemas se compone de entidades, relaciones, atributos, dominios de atributos e identificadores. El esquema conceptual también tendrá una documentación, que se irá produciendo durante su desarrollo. Las tareas a realizar en el diseño conceptual son las siguientes:

1. Identificar las entidades.
2. Identificar las relaciones.
3. Identificar los atributos y asociarlos a entidades y relaciones.
4. Determinar los dominios de los atributos.
5. Determinar los identificadores.
6. Determinar las jerarquías de generalización (si las hay).
7. Dibujar el diagrama entidad-relación.
8. Revisar el esquema conceptual local con el usuario.

**Tarea 1: Identificar las entidades**

En primer lugar hay que definir los principales objetos que interesan al usuario. Estos objetos serán las entidades. Una forma de identificar las entidades es examinar las especificaciones de requisitos de usuario. En estas especificaciones se buscan los nombres (por ejemplo: número de empleado, nombre de empleado, número de inmueble, dirección del inmueble, alquiler, número de habitaciones). También se buscan objetos importantes como personas, lugares o conceptos de interés, excluyendo aquellos nombres que sólo son propiedades de otros objetos. Por ejemplo, se pueden agrupar el número de empleado y el nombre de empleado en una entidad denominada *empleado*, y agrupar número de inmueble, dirección del inmueble, alquiler y número de habitaciones en otra entidad denominada *inmueble*. Otra forma de identificar las entidades es buscar aquellos objetos que existen por sí mismos. Por ejemplo, *empleado* es una entidad porque los empleados existen, sepamos o no sus nombres, direcciones y teléfonos. Siempre que sea posible, el usuario debe colaborar en la identificación de las entidades. A veces, es difícil identificar las entidades por la forma en que aparecen en las especificaciones de requisitos. Los usuarios, a veces, hablan utilizando ejemplos o analogías. En lugar de hablar de empleados en general, hablan de personas concretas, o bien, hablan de los puestos que ocupan esas personas. No siempre es obvio saber si un objeto es una entidad, una relación o un atributo. Los diseñadores de bases de datos deben tener una visión selectiva y clasificar las cosas que observan dentro del contexto de la empresa u organización. A partir de unas especificaciones de usuario es posible que no se pueda deducir un conjunto único de entidades, pero después de varias iteraciones del proceso de análisis, se llegará a obtener un conjunto de entidades que sean adecuadas para el sistema que se ha de construir. Conforme se van identificando las entidades, se les dan nombres que tengan un significado y que sean obvias para el usuario. Los nombres de las entidades y sus descripciones se anotan en el diccionario de datos. Cuando sea posible, se debe anotar también el número aproximado de ocurrencias de cada entidad. Si una entidad se conoce por varios nombres, éstos se deben anotar en el diccionario de datos como alias o sinónimos.

**Tarea 2: Identificar las relaciones**

Una vez definidas las entidades, se deben definir las relaciones existentes entre ellas. Del mismo modo que para identificar las entidades se buscaban nombres en las especificaciones de requisitos, para identificar las relaciones se suelen buscar las expresiones verbales (por ejemplo: oficina tiene empleados, empleado gestiona inmueble, cliente visita inmueble). Si las especificaciones de requisitos reflejan estas relaciones es porque son importantes para la empresa y, por lo tanto, se deben reflejar en el esquema conceptual. Pero sólo interesan las relaciones que son necesarias. La mayoría de las relaciones son binarias (entre dos entidades), pero no hay que olvidar que también puede haber relaciones en las que participen más de dos entidades, así como relaciones recursivas. Es muy importante repasar las especificaciones para comprobar que todas las relaciones, explícitas o implícitas, se han encontrado. Si se tienen pocas entidades, se puede comprobar por parejas si hay alguna relación entre ellas. De todos modos, las relaciones que no se identifican ahora se suelen encontrar cuando se valida el esquema con las transacciones que debe soportar. Una vez identificadas todas las relaciones, hay que determinar la cardinalidad mínima y máxima con la que participa cada entidad en cada una de ellas. De este modo, el esquema representa de un modo más explícito la semántica de las relaciones. La cardinalidad es un tipo de restricción que se utiliza para comprobar y mantener la calidad de los datos. Estas restricciones son aserciones sobre las entidades que se pueden aplicar cuando se actualiza la base de datos para determinar si las actualizaciones violan o no las reglas establecidas sobre la semántica de los datos. Conforme se van identificando las relaciones, se les van asignando nombres que tengan significado para el usuario. En el diccionario de datos se anotan los nombres de las relaciones, su descripción y las cardinalidades con las que participan las entidades en ellas.

**Tarea 3: Identificar los atributos y asociarlos a entidades y**

**Relaciones**

Al igual que con las entidades, se buscan nombres en las especificaciones de requisitos. Son atributos los nombres que identifican propiedades, cualidades, identificadores o características de entidades o relaciones. Lo más sencillo es preguntarse, para cada entidad y cada relación, ¿qué información se quiere saber de..? La respuesta a esta pregunta se debe encontrar en las especificaciones de requisitos. Pero, en ocasiones, será necesario preguntar a los usuarios para que aclaren los requisitos. Desgraciadamente, los usuarios pueden dar respuestas a esta pregunta que también contengan otros conceptos, por lo que hay que considerar sus respuestas con mucho cuidado. Al identificar los atributos, hay que tener en cuenta si son simples o compuestos. Por ejemplo, el atributo *dirección* puede ser simple, teniendo la dirección completa como un solo valor: `Av. Orellana 45, Guayaquil; o puede ser un atributo compuesto, formado por la *calle* (`Av. Orellana'), el *número* (`45') y la *población* (`Guayaquil'). El escoger entre atributo simple o compuesto depende de los requisitos del usuario. Si el usuario no necesita acceder a cada uno de los componentes de la dirección por separado, se puede representar como un atributo simple. Pero si el usuario quiere acceder a los componentes de forma individual, entonces se debe representar como un atributo compuesto. También se deben identificar los atributos derivados o calculados, que son aquellos cuyo valor se puede calcular a partir de los valores de otros atributos. Por ejemplo, el número de empleados de cada oficina, la edad de los empleados o el número de inmuebles que gestiona cada empleado. Algunos diseñadores no representan los atributos derivados en los esquemas conceptuales. Si se hace, se debe indicar claramente que el atributo es derivado y a partir de qué atributos se obtiene su valor. Donde hay que considerar los atributos derivados es en el diseño físico. Cuando se están identificando los atributos, se puede descubrir alguna entidad que no se ha identificado previamente, por lo que hay que volver al principio introduciendo esta entidad y viendo si se relaciona con otras entidades. Es muy útil elaborar una lista de atributos e ir eliminándolos de la lista conforme se vayan asociando a una entidad o relación. De este modo, uno se puede asegurar de que cada atributo se asocia a una sola entidad o relación, y que cuando la lista se ha acabado, se han asociado todos los atributos. Hay que tener mucho cuidado cuando parece que un mismo atributo se debe asociar a varias entidades. Esto puede ser por una de las siguientes causas:

* Se han identificado varias entidades, como *director*, *supervisor* y *administrativo*, cuando, de hecho, pueden representarse como una sola entidad denominada *empleado*. En este caso, se puede escoger entre introducir una jerarquía de generalización, o dejar las entidades que representan cada uno de los puestos de empleado.
* Se ha identificado una relación entre entidades. En este caso, se debe asociar el atributo a una sola de las entidades y hay que asegurarse de que la relación ya se había identificado previamente. Si no es así, se debe actualizar la documentación para recoger la nueva relación.

Conforme se van identificando los atributos, se les asignan nombres que tengan significado para el usuario. De cada atributo se debe anotar la siguiente información:

* Nombre y descripción del atributo.
* Alias o sinónimos por los que se conoce al atributo.
* Tipo de dato y longitud.
* Valores por defecto del atributo (si se especifican).
* Si el atributo siempre va a tener un valor (si admite o no nulos).
* Si el atributo es compuesto y, en su caso, qué atributos simples lo forman.
* Si el atributo es derivado y, en su caso, cómo se calcula su valor.
* Si el atributo es multievaluado.

**Tarea 4: Determinar los dominios de los atributos**

El dominio de un atributo es el conjunto de valores que puede tomar el atributo. Por ejemplo el dominio de los números de oficina son las tiras de hasta tres caracteres en donde el primero es una letra y el siguiente o los dos siguientes son dígitos en el rango de 1 a 99; el dominio de los números de teléfono y los números de fax son las tiras de 9 dígitos. Un esquema conceptual está completo si incluye los dominios de cada atributo: los valores permitidos para cada atributo, su tamaño y su formato. También se puede incluir información adicional sobre los dominios como, por ejemplo, las operaciones que se pueden realizar sobre cada atributo, qué atributos pueden compararse entre sí o qué atributos pueden combinarse con otros. Aunque sería muy interesante que el sistema final respetara todas estas indicaciones sobre los dominios, esto es todavía una línea abierta de investigación. Toda la información sobre los dominios se debe anotar también en el diccionario de datos.

**Tarea 5: Determinar los identificadores**

Cada entidad tiene al menos un identificador. En este paso, se trata de encontrar todos los identificadores de cada una de las entidades. Los identificadores pueden ser simples o compuestos. De cada entidad se escogerá uno de los identificadores como clave primaria en la fase del diseño lógico. Cuando se determinan los identificadores es fácil darse cuenta de si una entidad es fuerte o débil. Si una entidad tiene al menos un identificador, es *fuerte* (otras denominaciones son *padre*, *propietaria* o *dominante*). Si una entidad no tiene atributos que le sirvan de identificador, es *débil* (otras denominaciones son *hijo*, *dependiente* o *subordinada*). Todos los identificadores de las entidades se deben anotar en el diccionario de datos.

**Tarea 6: Determinar las jerarquías de generalización**

En este paso hay que observar las entidades que se han identificado hasta el momento. Hay que ver si es necesario reflejar las diferencias entre distintas ocurrencias de una entidad, con lo que surgirán nuevas sub-entidades de esta entidad genérica; o bien, si hay entidades que tienen características en común y que realmente son sub-entidades de una nueva entidad genérica. En cada jerarquía hay que determinar si es total o parcial y exclusiva o superpuesta.

**Tarea 7: Dibujar el diagrama entidad-relación**

Una vez identificados todos los conceptos, se puede dibujar el diagrama entidad-relación correspondiente a una de las vistas de los usuarios. Se obtiene así un esquema conceptual local.

**Tarea 8: Revisar el esquema conceptual local con el usuario**

Antes de dar por finalizada la fase del diseño conceptual, se debe revisar el esquema conceptual local con el usuario. Este esquema está formado por el diagrama entidad-relación y toda la documentación que describe el esquema. Si se encuentra alguna anomalía, hay que corregirla haciendo los cambios oportunos, por lo que posiblemente haya que repetir alguno de los pasos anteriores. Este proceso debe repetirse hasta que se esté seguro de que el esquema conceptual es una fiel representación de la parte de la empresa que se está tratando de modelar.

**2.2.1.1 El modelo Entidad – Relación**

El modelo entidad-relación es el modelo conceptual más utilizado para el diseño conceptual de bases de datos. Fue introducido por Peter Chen en 1976. El modelo entidad-relación está formado por un conjunto de conceptos que permiten describir la realidad mediante un conjunto de representaciones gráficas y lingüísticas. Originalmente, el modelo entidad-relación sólo incluía los conceptos de entidad, relación y atributo. Más tarde, se añadieron otros conceptos, como los atributos compuestos y las jerarquías de generalización, en lo que se ha denominado *modelo entidad-relación extendido*.

**Gráfico 2**

\begin{figure}
\centerline {\hbox{\psfig{figure=modelo_er.ps}}}\end{figure}

**Conceptos del modelo entidad – relación extendido**

**Entidad**

Cualquier tipo de objeto o concepto sobre el que se recoge información: cosa, persona, concepto abstracto o suceso. Por ejemplo: coches, casas, empleados, clientes, empresas, oficios, diseños de productos, conciertos, excursiones, etc. Las entidades se representan gráficamente mediante rectángulos y su nombre aparece en el interior. Un nombre de entidad sólo puede aparecer una vez en el esquema conceptual. Hay dos tipos de entidades: fuertes y débiles. Una *entidad débil* es una entidad cuya existencia depende de la existencia de otra entidad. Una *entidad fuerte* es una entidad que no es débil.

**Relación (interrelación)**

Es una correspondencia o asociación entre dos o más entidades. Cada relación tiene un nombre que describe su función. Las relaciones se representan gráficamente mediante rombos y su nombre aparece en el interior. Las entidades que están involucradas en una determinada relación se denominan *entidades participantes*. El número de participantes en una relación es lo que se denomina *grado* de la relación. Por lo tanto, una relación en la que participan dos entidades es una relación *binaria*; si son tres las entidades participantes, la relación es *ternaria*; etc. Una *relación recursiva* es una relación donde la misma entidad participa más de una vez en la relación con distintos papeles. El nombre de estos papeles es importante para determinar la función de cada participación. La *cardinalidad* con la que una entidad participa en una relación especifica el número mínimo y el número máximo de correspondencias en las que puede tomar parte cada ocurrencia de dicha entidad. La participación de una entidad en una relación es *obligatoria (total)* si la existencia de cada una de sus ocurrencias requiere la existencia de, al menos, una ocurrencia de la otra entidad participante. Si no, la participación es *opcional (parcial)*. Las reglas que definen la cardinalidad de las relaciones son las *reglas de negocio*. A veces, surgen problemas cuando se está diseñado un esquema conceptual. Estos problemas, denominados *trampas*, suelen producirse a causa de una mala interpretación en el significado de alguna relación, por lo que es importante comprobar que el esquema conceptual carece de dichas trampas. En general, para encontrar las trampas, hay que asegurarse de que se entiende completamente el significado de cada relación. Si no se entienden las relaciones, se puede crear un esquema que no represente fielmente la realidad. Una de las trampas que pueden encontrarse ocurre cuando el esquema representa una relación entre entidades, pero el camino entre algunas de sus ocurrencias es ambiguo. El modo de resolverla es reestructurando el esquema para representar la asociación entre las entidades correctamente. Otra de las trampas sucede cuando un esquema sugiere la existencia de una relación entre entidades, pero el camino entre una y otra no existe para algunas de sus ocurrencias. En este caso, se produce una pérdida de información que se puede subsanar introduciendo la relación que sugería el esquema y que no estaba representada.

**Atributo**

Es una característica de interés o un hecho sobre una entidad o sobre una relación. Los atributos representan las propiedades básicas de las entidades y de las relaciones. Toda la información extensiva es portada por los atributos. Gráficamente, se representan mediante bolitas que cuelgan de las entidades o relaciones a las que pertenecen. Cada atributo tiene un conjunto de valores asociados denominado *dominio*. El dominio define todos los valores posibles que puede tomar un atributo. Puede haber varios atributos definidos sobre un mismo dominio. Los atributos pueden ser simples o compuestos. Un *atributo simple* es un atributo que tiene un solo componente, que no se puede dividir en partes más pequeñas que tengan un significado propio. Un *atributo compuesto* es un atributo con varios componentes, cada uno con un significado por sí mismo. Un grupo de atributos se representa mediante un atributo compuesto cuando tienen afinidad en cuanto a su significado, o en cuanto a su uso. Un atributo compuesto se representa gráficamente mediante un óvalo. Los atributos también pueden clasificarse en monovalentes o polivalentes. Un *atributo monovalente* es aquel que tiene un solo valor para cada ocurrencia de la entidad o relación a la que pertenece. Un *atributo polivalente* es aquel que tiene varios valores para cada ocurrencia de la entidad o relación a la que pertenece. A estos atributos también se les denomina *multivaluados*, y pueden tener un número máximo y un número mínimo de valores. La *cardinalidad* de un atributo indica el número mínimo y el número máximo de valores que puede tomar para cada ocurrencia de la entidad o relación a la que pertenece. Por último, los atributos pueden ser derivados. Un *atributo derivado* es aquel que representa un valor que se puede obtener a partir del valor de uno o varios atributos, que no necesariamente deben pertenecer a la misma entidad o relación.

**Identificador**

Un identificador de una entidad es un atributo o conjunto de atributos que determina de modo único cada ocurrencia de esa entidad. Un identificador de una entidad debe cumplir dos condiciones:

1. No pueden existir dos ocurrencias de la entidad con el mismo valor del identificador.
2. Si se omite cualquier atributo del identificador, la condición anterior deja de cumplirse.

Toda entidad tiene al menos un identificador y puede tener varios identificadores alternativos. Las relaciones no tienen identificadores.

**Jerarquía de generalización**

Una entidad *E* es una generalización de un grupo de entidades *E$_1$, E$_2$, ... E$_n$*, si cada ocurrencia de cada una de esas entidades es también una ocurrencia de *E*. Todas las propiedades de la entidad genérica *E* son heredadas por las subentidades. Cada jerarquía es total o parcial, y exclusiva o superpuesta. Una jerarquía es *total* si cada ocurrencia de la entidad genérica corresponde al menos con una ocurrencia de alguna sub-entidad. Es *parcial* si existe alguna ocurrencia de la entidad genérica que no corresponde con ninguna ocurrencia de ninguna sub-entidad. Una jerarquía es *exclusiva* si cada ocurrencia de la entidad genérica corresponde, como mucho, con una ocurrencia de una sola de las sub-entidades. Es *superpuesta* si existe alguna ocurrencia de la entidad genérica que corresponde a ocurrencias de dos o más subentidades diferentes. Un *subconjunto* es un caso particular de generalización con una sola entidad como sub-entidad. Un subconjunto siempre es una jerarquía parcial y exclusiva.

**2.2.2 Metodología del diseño lógico**

La metodología que se va a seguir para el diseño lógico en el modelo relacional consta de dos fases, cada una de ellas compuesta por varios pasos que se detallan a continuación.

* + - 1. **Construir y validar los esquemas lógicos locales para cada vista de usuario.**
* Convertir los esquemas conceptuales locales en esquemas lógicos locales.
* Derivar un conjunto de relaciones (tablas) para cada esquema lógico local.
* Validar cada esquema mediante la normalización.
* Validar cada esquema frente a las transacciones del usuario.
* Dibujar el diagrama entidad-relación.
* Definir las restricciones de integridad.
* Revisar cada esquema lógico local con el usuario correspondiente.

**2.2.2.2 Construir y validar el esquema lógico global.**

* Mezclar los esquemas lógicos locales en un esquema lógico global.
* Validar el esquema lógico global.
* Estudiar el crecimiento futuro.
* Dibujar el diagrama entidad-relación final.
* Revisar el esquema lógico global con los usuarios.

En la primera fase, se construyen los esquemas lógicos locales para cada vista de usuario y se validan. En esta fase se refinan los esquemas conceptuales creados durante el diseño conceptual, eliminando las estructuras de datos que no se pueden implementar de manera directa sobre el modelo que soporta el SGBD (sistema generador de base de datos), en el caso que nos ocupa, el modelo relacional. Una vez hecho esto, se obtiene un primer esquema lógico que se valida mediante la normalización y frente a las transacciones que el sistema debe llevar a cabo, tal y como se refleja en las especificaciones de requisitos de usuario. El esquema lógico ya validado se puede utilizar como base para el desarrollo de prototipos. Una vez finalizada esta fase, se dispone de un esquema lógico para cada vista de usuario que es correcto, comprensible y sin ambigüedad.

**2.2.2.1.1 Convertir los esquemas conceptuales locales en esquemas lógicos locales**

En este paso, se eliminan de cada esquema conceptual las estructuras de datos que los sistemas relacionales no modelan directamente:

**(a)**

*Eliminar las relaciones de muchos a muchos,* sustituyendo cada una de ellas por una nueva entidad intermedia y dos relaciones de uno a muchos de esta nueva entidad con las entidades originales. La nueva entidad será débil, ya que sus ocurrencias dependen de la existencia de ocurrencias en las entidades originales.

**(b)**

*Eliminar las relaciones entre tres o más entidades,* sustituyendo cada una de ellas por una nueva entidad (débil) intermedia que se relaciona con cada una de las entidades originales. La cardinalidad de estas nuevas relaciones binarias dependerá de su significado.

**(c)**

*Eliminar las relaciones recursivas,* sustituyendo cada una de ellas por una nueva entidad (débil) y dos relaciones binarias de esta nueva entidad con la entidad original. La cardinalidad de estas relaciones dependerá de su significado.

**(d)**

*Eliminar las relaciones con atributos,* sustituyendo cada una de ellas por una nueva entidad (débil) y las relaciones binarias correspondientes de esta nueva entidad con las entidades originales. La cardinalidad de estas relaciones dependerá del tipo de la relación original y de su significado.

**(e)**

*Eliminar los atributos multievaluados,* sustituyendo cada uno de ellos por una nueva entidad (débil) y una relación binaria de uno a muchos con la entidad original.

**(f)**

*Revisar las relaciones de uno a uno*, ya que es posible que se hayan identificado dos entidades que representen el mismo objeto (sinónimos). Si así fuera, ambas entidades deben integrarse en una sola.

**(g)**

*Eliminar las relaciones redundantes*. Una relación es redundante cuando se puede obtener la misma información que ella aporta mediante otras relaciones. El hecho de que haya dos caminos diferentes entre dos entidades no implica que uno de los caminos corresponda a una relación redundante, eso dependerá del significado de cada relación.

Una vez finalizado este paso, es más correcto referirse a los esquemas conceptuales locales refinados como esquemas lógicos locales, ya que se adaptan al modelo de base de datos que soporta el SGBD escogido.

**2.2.2.1.2 Derivar un conjunto de relaciones (tablas) para cada esquema lógico local**

En este paso, se obtiene un conjunto de relaciones (tablas) para cada uno de los esquemas lógicos locales en donde se representen las entidades y relaciones entre entidades, que se describen en cada una de las vistas que los usuarios tienen de la empresa. Cada relación de la base de datos tendrá un nombre, y el nombre de sus atributos aparecerá, a continuación, entre paréntesis. El atributo o atributos que forman la clave primaria se subrayan. Las claves ajenas, mecanismo que se utiliza para representar las relaciones entre entidades en el modelo relacional, se especifican aparte indicando la relación (tabla) a la que hacen referencia. A continuación, se describe cómo las relaciones (tablas) del modelo relacional representan las entidades y relaciones que pueden aparecer en los esquemas lógicos.

**(a)**

*Entidades fuertes.* Crear una relación para cada entidad fuerte que incluya todos sus atributos simples. De los atributos compuestos incluir sólo sus componentes.

Cada uno de los identificadores de la entidad será una clave candidata. De entre las claves candidatas hay que escoger la clave primaria; el resto serán claves alternativas. Para escoger la clave primaria entre las claves candidatas se pueden seguir estas indicaciones:

* Escoger la clave candidata que tenga menos atributos.
* Escoger la clave candidata cuyos valores no tengan probabilidad de cambiar en el futuro.
* Escoger la clave candidata cuyos valores no tengan probabilidad de perder la unicidad en el futuro.
* Escoger la clave candidata con el mínimo número de caracteres (si es de tipo texto).
* Escoger la clave candidata más fácil de utilizar desde el punto de vista de los usuarios.

**(b)**

*Entidades débiles.* Crear una relación para cada entidad débil incluyendo todos sus atributos simples. De los atributos compuestos incluir sólo sus componentes. Añadir una clave ajena a la entidad de la que depende. Para ello, se incluye la clave primaria de la relación que representa a la entidad padre en la nueva relación creada para la entidad débil. A continuación, determinar la clave primaria de la nueva relación.

**(c)**

*Relaciones binarias de uno a uno.* Para cada relación binaria se incluyen los atributos de la clave primaria de la entidad padre en la relación (tabla) que representa a la entidad hijo, para actuar como una clave ajena. La entidad hijo es la que participa de forma total (obligatoria) en la relación, mientras que la entidad padre es la que participa de forma parcial (opcional). Si las dos entidades participan de forma total o parcial en la relación, la elección de padre e hijo es arbitraria. Además, en caso de que ambas entidades participen de forma total en la relación, se tiene la opción de integrar las dos entidades en una sola relación (tabla). Esto se suele hacer si una de las entidades no participa en ninguna otra relación.

**(d)**

*Relaciones binarias de uno a muchos.* Como en las relaciones de uno a uno, se incluyen los atributos de la clave primaria de la entidad padre en la relación (tabla) que representa a la entidad hijo, para actuar como una clave ajena. Pero ahora, la entidad padre es la de ``la parte del muchos'' (cada padre tiene muchos hijos), mientras que la entidad hijo es la de la parte del uno (cada hijo tiene un solo padre).

**(e)**

*Jerarquías de generalización.* En las jerarquías, se denomina entidad padre a la entidad genérica y entidades hijo a las subentidades. Hay tres opciones distintas para representar las jerarquías. La elección de la más adecuada se hará en función de su tipo (total/parcial, exclusiva/superpuesta).

1. Crear una relación por cada entidad. Las relaciones de las entidades hijo heredan como clave primaria la de la entidad padre. Por lo tanto, la clave primaria de las entidades hijo es también una clave ajena al padre. Esta opción sirve para cualquier tipo de jerarquía, total o parcial y exclusiva o superpuesta.
2. Crear una relación por cada entidad hijo, heredando los atributos de la entidad padre. Esta opción sólo sirve para jerarquías totales y exclusivas.
3. Integrar todas las entidades en una relación, incluyendo en ella los atributos de la entidad padre, los atributos de todos los hijos y un atributo discriminativo para indicar el caso al cual pertenece la entidad en consideración. Esta opción sirve para cualquier tipo de jerarquía. Si la jerarquía es superpuesta, el atributo discriminativo será multievaluado.

Una vez obtenidas las relaciones con sus atributos, claves primarias y claves ajenas, sólo queda actualizar el diccionario de datos con los nuevos atributos que se hayan identificado en este paso.

**2.2.2.1.3 Validar cada esquema mediante la normalización**

La normalización se utiliza para mejorar el esquema lógico, de modo que satisfaga ciertas restricciones que eviten la duplicidad de datos. La normalización garantiza que el esquema resultante se encuentra más próximo al modelo de la empresa, que es consistente y que tiene la mínima redundancia y la máxima estabilidad. La normalización es un proceso que permite decidir a qué entidad pertenece cada atributo. Uno de los conceptos básicos del modelo relacional es que los atributos se agrupan en relaciones (tablas) porque están relacionados a nivel lógico. En la mayoría de las ocasiones, una base de datos normalizada no proporciona la máxima eficiencia, sin embargo, el objetivo ahora es conseguir una base de datos normalizada por las siguientes razones:

* Un esquema normalizado organiza los datos de acuerdo a sus dependencias funcionales, es decir, de acuerdo a sus relaciones lógicas.
* El esquema lógico no tiene porqué ser el esquema final. Debe representar lo que el diseñador entiende sobre la naturaleza y el significado de los datos de la empresa. Si se establecen unos objetivos en cuanto a prestaciones, el diseño físico cambiará el esquema lógico de modo adecuado. Una posibilidad es que algunas relaciones normalizadas se desnormalicen. Pero la desnormalización no implica que se haya malgastado tiempo normalizando, ya que mediante este proceso el diseñador aprende más sobre el significado de los datos. De hecho, la normalización obliga a entender completamente cada uno de los atributos que se han de representar en la base de datos.
* Un esquema normalizado es robusto y carece de redundancias, por lo que está libre de ciertas anomalías que éstas pueden provocar cuando se actualiza la base de datos.
* Los equipos informáticos de hoy en día son mucho más potentes, por lo que en ocasiones es más razonable implementar bases de datos fáciles de manejar (las normalizadas), a costa de un tiempo adicional de proceso.
* La normalización produce bases de datos con esquemas flexibles que pueden extenderse con facilidad.

**2.2.2.1.4 Validar cada esquema frente a las transacciones del usuario**

El objetivo de este paso es validar cada esquema lógico local para garantizar que puede soportar las transacciones requeridas por los correspondientes usuarios. Estas transacciones se encontrarán en las especificaciones de requisitos de usuario. Lo que se debe hacer es tratar de realizar las transacciones de forma manual utilizando el diagrama entidad-relación, el diccionario de datos y las conexiones que establecen las claves ajenas de las relaciones (tablas). Si todas las transacciones se pueden realizar, el esquema queda validado. Pero si alguna transacción no se puede realizar, seguramente será porque alguna entidad, relación o atributo no se ha incluido en el esquema.

**2.2.2.1.5 Dibujar el diagrama entidad-relación**

En este momento, se puede dibujar el diagrama entidad-relación final para cada vista de usuario que recoja la representación lógica de los datos desde su punto de vista. Este diagrama habrá sido validado mediante la normalización y frente a las transacciones de los usuarios.

**2.2.2.1.6 Definir las restricciones de integridad**

Las restricciones de integridad son reglas que se quieren imponer para proteger la base de datos, de modo que no pueda llegar a un estado inconsistente. Hay cinco tipos de restricciones de integridad.

**(a)**

*Datos requeridos.* Algunos atributos deben contener valores en todo momento, es decir, no admiten nulos.

**(b)**

*Restricciones de dominios.* Todos los atributos tienen un dominio asociado, que es el conjunto de los valores que cada atributo puede tomar.

**(c)**

*Integridad de entidades.*  El identificador de una entidad no puede ser nulo, por lo tanto, las claves primarias de las relaciones (tablas) no admiten nulos.

**(d)**

*Integridad referencial.* Es cuando una clave ajena enlaza cada tupla de la relación hijo con la tupla de la relación padre que tiene el mismo valor en su clave primaria. La integridad referencial dice que si una clave ajena tiene un valor (si es no nula), ese valor debe ser uno de los valores de la clave primaria a la que referencia. Hay varios aspectos a tener en cuenta sobre las claves ajenas para lograr que se cumpla la integridad referencial.

1. ¿Admite nulos la clave ajena? Cada clave ajena expresa una relación. Si la participación de la entidad hijo en la relación es total, entonces la clave ajena no admite nulos; si es parcial, la clave ajena debe aceptar nulos.
2. ¿Qué hacer cuando se quiere borrar una ocurrencia de la entidad padre que tiene algún hijo? O lo que es lo mismo, ¿qué hacer cuando se quiere borrar una tupla que está siendo referenciada por otra tupla a través de una clave ajena? Hay varias respuestas posibles:
   * *Restringir:* no se pueden borrar tuplas que están siendo referenciadas por otras tuplas.
   * *Propagar:* se borra la tupla deseada y se propaga el borrado a todas las tuplas que le hacen referencia.
   * *Anular:* se borra la tupla deseada y todas las referencias que tenía se ponen, automáticamente, a nulo (esta respuesta sólo es válida si la clave ajena acepta nulos).
   * *Valor por defecto:* se borra la tupla deseada y todas las referencias toman, automáticamente, el valor por defecto (esta respuesta sólo es válida si se ha especificado un valor por defecto para la clave ajena).
   * *No comprobar:* se borra la tupla deseada y no se hace nada para garantizar que se sigue cumpliendo la integridad referencial.
3. ¿Qué hacer cuando se quiere modificar la clave primaria de una tupla que está siendo referenciada por otra tupla a través de una clave ajena? Las respuestas posibles son las mismas que en el caso anterior. Cuando se escoge propagar, se actualiza la clave primaria en la tupla deseada y se propaga el cambio a los valores de clave ajena que le hacían referencia.

**(e)**

*Reglas de negocio.* Cualquier operación que se realice sobre los datos debe cumplir las restricciones que impone el funcionamiento de la empresa.

Todas las restricciones de integridad establecidas en este paso se deben reflejar en el diccionario de datos para que puedan ser tenidas en cuenta durante la fase del diseño físico.

**2.2.2.1.7 Revisar cada esquema lógico local con el Usuario**

**correspondiente**

Para garantizar que cada esquema lógico local es una fiel representación de la vista del usuario lo que se debe hacer es comprobar con él que lo reflejado en el esquema y en la documentación es correcto y está completo.

**2.2.2.1.8 Mezclar los esquemas lógicos locales en un**

**Esquema lógico**

En este paso, se deben integrar todos los esquemas locales en un solo esquema global. En un sistema pequeño, con dos o tres vistas de usuario y unas pocas entidades y relaciones, es relativamente sencillo comparar los esquemas locales, mezclarlos y resolver cualquier tipo de diferencia que pueda existir. Pero en los sistemas grandes, se debe seguir un proceso más sistemático para llevar a cabo este paso con éxito:

1. Revisar los nombres de las entidades y sus claves primarias.
2. Revisar los nombres de las relaciones.
3. Mezclar las entidades de las vistas locales.
4. Incluir (sin mezclar) las entidades que pertenecen a una sola vista de usuario.
5. Mezclar las relaciones de las vistas locales.
6. Incluir (sin mezclar) las relaciones que pertenecen a una sola vista de usuario.
7. Comprobar que no se ha omitido ninguna entidad ni relación.
8. Comprobar las claves ajenas.
9. Comprobar las restricciones de integridad.
10. Dibujar el esquema lógico global.
11. Actualizar la documentación.

**2.2.2.1.9 Validar el esquema lógico global**

Este proceso de validación se realiza, de nuevo, mediante la normalización y mediante la prueba frente a las transacciones de los usuarios. Pero ahora sólo hay que normalizar las relaciones que hayan cambiado al mezclar los esquemas lógicos locales y sólo hay que probar las transacciones que requieran acceso a áreas que hayan sufrido algún cambio.

**2.2.2.1.10 Estudiar el crecimiento futuro**

En este paso, se trata de comprobar que el esquema obtenido puede acomodar los futuros cambios en los requisitos con un impacto mínimo. Si el esquema lógico se puede extender fácilmente, cualquiera de los cambios previstos se podrá incorporar al mismo con un efecto mínimo sobre los usuarios existentes.

**2.2.2.1.11 Dibujar el diagrama entidad-relación final**

Una vez validado el esquema lógico global, ya se puede dibujar el diagrama entidad-relación que representa el modelo de los datos de la empresa que son de interés. La documentación que describe este modelo (incluyendo el esquema relacional y el diccionario de datos) se debe actualizar y completar.

**2.2.2.1.12 Revisar el esquema lógico global con los usuarios**

Una vez más, se debe revisar con los usuarios el esquema global y la documentación obtenida para asegurarse de que son una fiel representación de la empresa.

**2.3 Lenguaje de Consultas**

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos. Pero como sucede con cualquier sistema de normalización hay excepciones para casi todo; de hecho, cada motor de bases de datos tiene sus peculiaridades y lo hace diferente de otro motor, por lo tanto, el lenguaje SQL normalizado (ANSI) no nos servirá para resolver todos los problemas, aunque si se puede asegurar que cualquier sentencia escrita en ANSI será interpretable por cualquier motor de datos.

**2.3.1 Historia del Lenguaje de Consultas**

La historia de SQL (que se pronuncia deletreando en inglés las letras que lo componen, es decir "ese-cu-ele" y no "siquel" como se oye a menudo) empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó SQL (sustancialmente adoptó el dialecto SQL de IBM) como estándar para los lenguajes relacionales y en 1987 se transformó en estándar ISO. Esta versión del estándar va con el nombre de SQL/86. En los años siguientes, éste ha sufrido diversas revisiones que han conducido primero a la versión SQL/89 y, posteriormente, a la actual SQL/92.

El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la intercomunicabilidad entre todos los productos que se basan en él. Desde el punto de vista práctico, por desgracia las cosas fueron de otro modo. Efectivamente, en general cada productor adopta e implementa en la propia base de datos sólo el corazón del lenguaje SQL (el así llamado Entry level o al máximo el Intermediate level), extendiéndolo de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

Actualmente, está en marcha un proceso de revisión del lenguaje por parte de los comités ANSI e ISO, que debería terminar en la definición de lo que en este momento se conoce como SQL3. Las características principales de esta nueva encarnación de SQL deberían ser su transformación en un lenguaje stand-alone (mientras ahora se usa como lenguaje hospedado en otros lenguajes) y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimediales.

**2.3.2 Componentes del SQL**

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

**2.3.2.1 Comandos**

Existen dos tipos de comandos SQL:

* Los DLL que permiten crear y definir nuevas bases de datos, campos e índices.
* Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Entre los comandos DLL tenemos a los siguientes:

**CREATE:** Utilizado para crear nuevas tablas, campos e índices.

**DROP:** Empleado para eliminar tablas e índices.

**ALTER:** Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

A continuación se presentan los comandos DML utilizados para el manejo de consultas:

**SELECT:** Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.

**INSERT:** Utilizado para cargar lotes de datos en la base de datos en una única operación.

**UPDATE:** Utilizado para modificar los valores de los campos y registros especificados.

**DELETE:** Utilizado para eliminar registros de una tabla de una base de datos.

**2.3.2.2 Cláusulas**

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular. Entre las cláusulas utilizadas para el manejo de consultas tenemos:

**FROM:** Utilizada para especificar la tabla de la cual se van a seleccionar los registros.

**WHERE:** Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.

**GROUP BY:** Utilizada para separar los registros seleccionados en grupos específicos.

**HAVING:** Utilizada para expresar la condición que debe satisfacer cada grupo.

**ORDER BY:** Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

**2.3.2.3 Operadores Lógicos**

Los operadores lógicos son utilizados para enlazar condiciones que se expresan en la cláusula **WHERE** o para negar una condición específica. A continuación se presentan a los operadores lógicos para manejo de consultas:

**AND:** Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.

**OR:** Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.

**NOT:** Negación lógica. Devuelve el valor contrario de la expresión.

**2.3.2.4 Operadores de Comparación**

Estos operadores son utilizados para realizar comparaciones entre valores o variables en las condiciones que se expresan en la cláusula **WHERE**. A continuación se presentan los operadores de comparación utilizados en el manejo de consultas:

**>** Mayor que

**<** Menor que

**>=** Mayor ó igual que

**<=** Menor ó igual que

**=** Igual que

**<>** Distinto que

**BETWEEN** Utilizado para especificar un rango de valores

**LIKE** Utilizado en la comparación de un modelo

**IN** Utilizado para especificar registros de una base de datos.

**2.3.2.5 Funciones de Agregado**

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros. A continuación se menciona las funciones de agregado utilizadas para el manejo de consultas:

**AVG:** Utilizada para calcular el promedio de los valores de un campo determinado.

**COUNT:** Utilizada para devolver el número de registros de la selección.

**SUM:** Utilizada para devolver la suma de todos los valores de un campo determinado.

**MAX:** Utilizada para devolver el valor más alto de un campo especificado.

**MIN:** Utilizada para devolver el valor más bajo de un campo especificado.

**2.3.3 Consultas de selección**

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

**2.3.3.1 Estructura básica de las consultas**

Para el manejo de consultas de bases de datos se debe seguir una estructura básica, que es la que especifica a continuación:

SELECT A1,A2,...,An

FROM r1,r2,...,rn

WHERE P

Donde Ai = atributo ( Campo de la tabla )

ri = relación ( Tabla )

P = predicado ( condición )

Por ejemplo, para seleccionar todos los nombres de las personas que tengan el apellido CASTRO de la tabla persona se utiliza una consulta como la siguiente:

SELECT nombre

FROM persona

WHERE apellido = " CASTRO"

Es posible renombrar los atributos y las relaciones, a veces por conveniencia y otras veces por ser necesario, para esto usamos la cláusula **AS**. A continuación se presenta el ejemplo anterior con el uso de esta cláusula.

SELECT P.nombre AS [PRIMER NOMBRE]

FROM persona P

WHERE apellido = "CASTRO"

La complejidad de una consulta puede aumentar cada vez más. Esto depende de los requerimientos que se haga y del uso de los comandos y cláusulas que se necesiten. Por ejemplo, puede haber **subconsultas**, que no son nada más que una consulta dentro de otra; es decir, un SELECT dentro de otro. Puede utilizar tres formas de sintaxis para crear una subconsulta:

comparación [ANY | ALL | SOME] (instrucción sql)

expresión [NOT] IN (instrucción sql)

[NOT] EXISTS (instrucción sql)

En donde:

**comparación**

Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.

**expresión**

Es una expresión por la que se busca el conjunto resultante de la subconsulta.

**instrucción sql**

Es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier otra instrucción SELECT. Debe ir entre paréntesis.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING. En una subconsulta, se utiliza una instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING.

Se puede utilizar el predicado ANY o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta. El ejemplo siguiente devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25 por ciento:

SELECT \* FROM Productos WHERE PrecioUnidad > ANY

(SELECT PrecioUnidad FROM DetallePedido WHERE Descuento >= 0 .25);

El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia ANY por ALL en el ejemplo anterior, la consulta devolverá únicamente aquellos productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25 por ciento. Esto es mucho más restrictivo.

El predicado IN se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El ejemplo siguiente devuelve todos los productos vendidos con un descuento igual o mayor al 25 por ciento:

SELECT \* FROM Productos WHERE IDProducto IN

(SELECT IDProducto FROM DetallePedido WHERE Descuento >= 0.25);

Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro. Supongamos que deseamos recuperar todos aquellos clientes que hayan realizado al menos un pedido:

SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE EXISTS

(SELECT FROM Pedidos WHERE Pedidos.IdPedido = Clientes.IdCliente)

Esta consulta es equivalente a esta otra:

SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE IdClientes IN

(SELECT Pedidos.IdCliente FROM Pedidos)

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM fuera de la subconsulta. El ejemplo siguiente devuelve los nombres de los empleados cuyo salario es igual o mayor que el salario medio de todos los empleados con el mismo título. A la tabla Empleados se le ha dado el alias T1:

SELECT Apellido, Nombre, Titulo, Salario FROM Empleados AS T1

WHERE Salario >= (SELECT Avg(Salario) FROM Empleados

WHERE T1.Titulo = Empleados.Titulo) ORDER BY Titulo;

**2.3.3.2 Consultas de combinación entre tablas**

Las vinculaciones entre tablas se realizan mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es la siguiente:

SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp tb2.campo2

En donde:

**tb1, tb2**: Son los nombres de las tablas desde las que se combinan los registros.

**campo1, campo2**: Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.

**Comp**: Es cualquier operador de comparación relacional: =, <, >, <=, >=, o <>.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones Equi son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar INNER JOIN con las tablas Departamentos y Empleados para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea LEFT JOIN o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso RIGHT JOIN.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador. El ejemplo siguiente muestra cómo podría combinar las tablas Categorías y Productos basándose en el campo IDCategoria:

SELECT Nombre\_Categoría, NombreProducto

FROM Categorias INNER JOIN Productos

ON Categorias.IDCategoria = Productos.IDCategoria;

**2.3.3.3 Consultas de autocombinación**

La autocombinación se utiliza para unir una tabla consigo misma, comparando valores de dos columnas con el mismo tipo de datos. La sintaxis es la siguiente:

SELECT alias1.columna, alias2.columna, ...

FROM tabla1 as alias1, tabla2 as alias2

WHERE alias1.columna = alias2.columna

AND otras condiciones

Por ejemplo, para visualizar el número, nombre y puesto de cada empleado, junto con el número, nombre y puesto del supervisor de cada uno de ellos se utilizaría la siguiente sentencia:

SELECT t.num\_emp, t.nombre, t.puesto, t.num\_sup,s.nombre, s.puesto

FROM empleados AS t, empleados AS s WHERE t.num\_sup = s.num\_emp

**2.3.3.4 Consultas de combinaciones no comunes**

La mayoría de las combinaciones están basadas en la igualdad de valores de las columnas que son el criterio de la combinación. Las no comunes se basan en otros operadores de combinación, tales como NOT, BETWEEN, <>, etc.

Por ejemplo, para listar el grado salarial, nombre, salario y puesto de cada empleado ordenando el resultado por grado y salario habría que ejecutar la siguiente sentencia:

SELECT grados.grado,empleados.nombre, empleados.salario, empleados.puesto

FROM empleados, grados

WHERE empleados.salario

BETWEEN grados.salarioinferior AND grados.salariosuperior

ORDER BY grados.grado, empleados.salario