La base de datos relacional

1. Introducción

En 1969, cuando se estaba utilizando bases de datos jerárquicas y en red, Edgar Codd desarrolló el modelo relacional, el modelo de bases de datos más usado hoy en día. El trabajo inicial de Codd "A Relational Model of Data for Large Shared Data Banks" fue publicado en Communications of the ACM en 1970. Su trabajo sobre normalización de bases de datos fue publicado como un informe técnico de IBM en 1971. Ocho años más tarde, en ACM Transactions of Database Systems, publicó varias extensiones al modelo relacional. En 1985 postuló una lista de 13 reglas que debía cumplir un producto de bases de datos para ser llamado relacional.

A finales de los setenta y principios de los ochenta se empezaron a desarrollar sistemas de bases de datos relacionales. Uno de los primeros fue el System R de IBM, que condujo al desarrollo de un lenguaje de consultas estructurado denominado SQL, que se ha convertido en el lenguaje estándar de los sistemas relacionales.

Le siguieron durante los años ochenta DB2 y SLQ/DS de IBM, y ORACLE de ORACLE Corporation.

Hoy en día, existen muchos SGBD relacionales, para todo tipo de ordenadores y plataformas, aunque muchos no son completamente fieles al modelo relacional.

Otros sistemas relacionales multiusuario son INGRES de Computer Associates, Informix de Informix Software Inc., Sybase de Sybase Inc, y SQL Server de Microsoft.

Ejemplos de sistemas relacionales de microordenadores son Paradox y dBase, Access de Microsoft, FoxPro publicado originalmente por Fox Software y posteriormente por Microsoft

Desde el sistema R de IBM a nuestros días han pasado 30 años y aún es el modelo dominante.

2. Objetivos de un SGBDR

Un sistema gestor de bases de datos relacional por definición debe cumplir los siguientes objetivos.

- **Independencia de los datos**

La independencia de los datos consiste en hacer que los programas no sean tan dependientes de la estructura de los datos.

Se han definido dos tipos de independencia:

La **independencia física** *PHYSICAL INDEPENDENCE*: consiste en poder modificar la definición interna de los datos (el esquema interno) sin que ello suponga una modificación de los programas existentes.

Por ejemplo, se puede cambiar la ubicación de la base de datos, o se puede añadir un índice sobre una tabla para que las consultas se ejecuten más rápidamente, sin que eso suponga una variación en los esquemas externos y conceptual, por lo que los programas (que utilizan el esquema externo) no se verán afectados.

La **independencia lógica** *LOGICAL INDEPENDENCE*: consiste en poder cambiar el esquema conceptual sin que ello suponga una modificación de los programas existentes.

Por ejemplo podemos añadir un nuevo dato en la tabla de clientes como la dirección de email sin que los esquemas externos se vean afectados.

- **Seguridad e integridad**

La **seguridad** *SECURITY* consiste en que los usuarios no puedan acceder a datos sin autorización.

Si juntamos toda la información de la empresa en un mismo sitio, el SGBD debe tener mecanismos para que cualquier usuario pueda tener acceso a únicamente la información que necesita para las tareas que tiene encomendadas.

Esta seguridad se consigue por medio de los esquemas externos, ya que el usuario sólo tiene acceso a su esquema externo que le proporciona los datos que el administrador ha considerado incluir en ese esquema. Para el usuario no habrá más datos que estos.

Además los SGBD tienen mecanismos para definir autorizaciones *AUTHORIZATIONS* que pueden ser de distinto tipo: autorización de lectura, de inserción, de actualización, autorizaciones especiales para poder variar el esquema conceptual etc.

La **integridad** *INTEGRITY* se refiere a que la información almacenada en la base de datos esté libre de errores. Esto no siempre es posible ya que existen distintos tipos de errores que tienen diferentes soluciones:

* Fallos de hardware. Estos errores no los puede evitar el SGBD pero se pueden subsanar facilitando copias de seguridad *BACKUPS* y procesos de recuperación *RECOVERY TASKS*.

* Fallos del programador. Puede que aparezcan datos erróneos en la base de datos como consecuencia de errores en el programa que genera estos datos. Para evitar al máximo este tipo de errores el sistema debe ser fácil de programar, cuantos más controles realice el sistema de forma automática, menos controles habrá que incluir a nivel de programación por lo que limitaremos la probabilidad de fallo y los programas deben ser probados con juegos de ensayos bien definidos.

* Fallos del usuario final. El usuario que introduce datos en la base de datos también puede cometer errores, el sistema debe permitir controlar al máximo la información que se introduce para limitar el número de estos errores, para ello los SGBD incluyen cláusulas de validación de los datos *DATA VALIDATION*, validaciones de diferentes tipos que veremos con más detalle más adelante.

* Fallos derivados de la concurrencia *DATA CONCURRENCY*. Ya que toda la información está centralizada y los distintos usuarios acceden a ella de forma simultánea, pueden ocurrir problemas cuando dos usuarios quieren acceder al mismo dato a la vez. Por ello el SGBD debe tener establecidos mecanismos para evitar este tipo de problema, bloqueos de registros *RECORDS LOCKS*, abortar automáticamente transacciones *AUTOMATIC TRANSACTION ABORTS*, etc.

- **Redundancia mínima**

La redundancia *REDUNDANCY* consiste en que exista algún dato repetido en varios lugares.

Por ejemplo si tenemos la dirección del cliente en la factura, en la cuenta contable, en los datos generales del cliente; esto como ya vimos anteriormente nos producirá varios problemas:

* la información repetida ocupa espacio innecesario.

* la variación de un domicilio supone el variar ese domicilio en todos los lugares donde esté almacenado => mayor tiempo de proceso

=> posibilidad de inconsistencia *INCONSISTENCY*

Por todo ello hay que evitar al máximo esa redundancia, esto se consigue utilizando herramientas de diseño *DESIGNING TOOLS* y obteniendo un diseño óptimo de la base de datos.

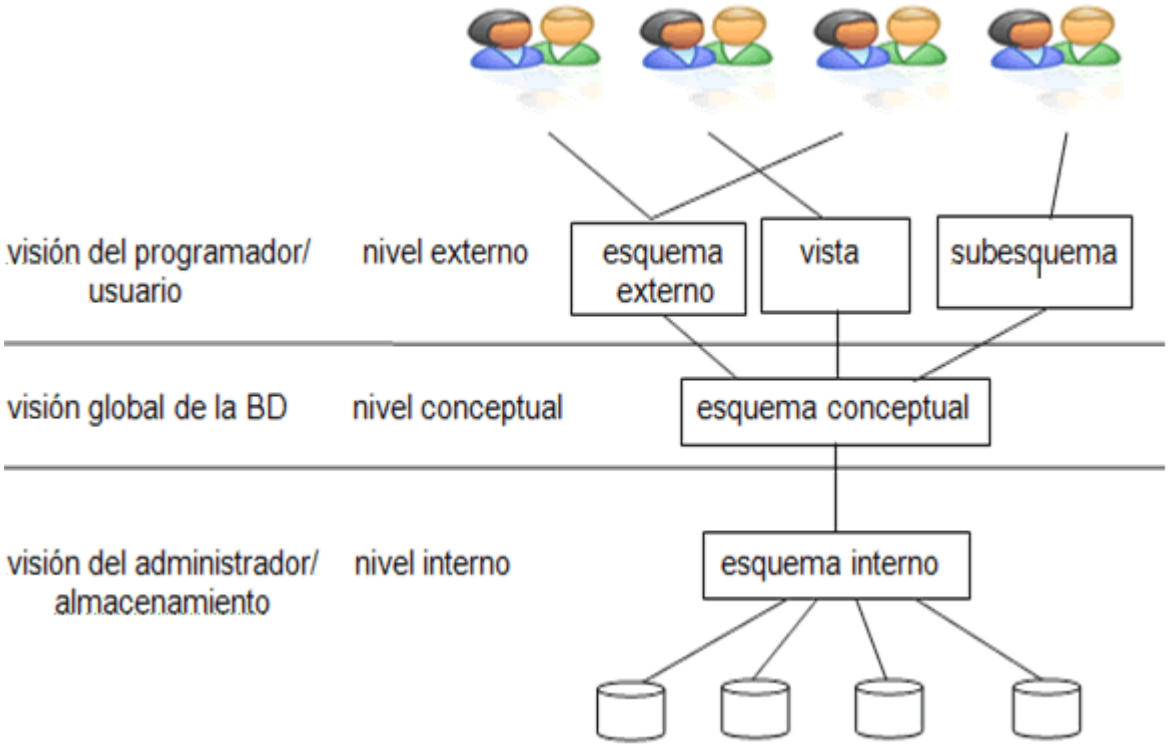
- **Facilidad de recuperación de la información**

Otro objetivo muy importante de un SGBD es el proporcionar al usuario (o programador) unas herramientas potentes de manejo de datos *POWERFUL TOOLS FOR DATA MANAGEMENT* para que pueda de manera sencilla y rápida, obtener toda la información que desea sin que, por ello se tenga que hacer un programa complejo.

Veremos que el SQL, lenguaje empleado para recuperar información de la base de datos, es un lenguaje muy potente y cercano al lenguaje hablado, y además los SGBD incluyen entornos gráficos *GRAFIC USER INTERFACES* sencillos de utilizar.

3. Arquitectura de un SGBD

Para lograr los objetivos anteriores se definió para la base de datos relacional una arquitectura estandarizada *STANDARDISED ARCHITECTURE*, en la cual los datos se pueden ver desde distintos puntos de vista. Esta arquitectura es la *ANSI/X3/SPARC* y define tres niveles o tres "visiones distintas de la base de datos", el nivel interno *INTERNAL LEVEL*, conceptual *CONCEPTUAL LEVEL* y externo *EXTERNAL LEVEL*.



3.1. El nivel interno

El nivel interno es el más cercano al almacenamiento de los datos, este nivel es el de la visión del administrador de la BD *DATABASE ADMINISTRATOR*, ya que es donde tenemos la definición física de los datos *PHYSICAL DEFINITION*, su organización interna *INTERNAL ORGANIZATION*, ubicación de la información en los distintos dispositivos *DEVICES*, tipos de datos *DATA TYPES*, definición de las reglas de validación *CHECK RULES*, controles de seguridad *SECURITY CHECKS*, etc...

3.2. El nivel conceptual

En este nivel se tiene una visión global de todos los datos que intervienen en la base de datos, pero a nivel de concepto olvidándose de la estructura interna. Podemos decir que en este nivel es donde tenemos la definición de qué datos se guardan en la base de datos y las relaciones que unen los datos entre sí.

3.3. El nivel externo

Este nivel representa la percepción de los datos por cada uno de los usuarios *USERS* o programadores, digamos que el usuario no ve todos los datos que aparecen en el nivel conceptual, sino sólo los que le hacen falta. Cada esquema externo *EXTERNAL SCHEMA* contiene la definición parcial del esquema conceptual *CONCEPTUAL SCHEMA*. De esta forma se puede proteger la información reservada a unos pocos usuarios. Para el usuario no habrá más datos en la base de datos que los definidos en los esquemas externos a los cuales pueda acceder.

El SGBD es el encargado de realizar el enlace entre los tres niveles de manera que cuando un usuario solicita un dato definido en un esquema externo, el SGBD lo busca en el lugar adecuado utilizando la definición interna de forma totalmente transparente para el usuario.

4. ELEMENTOS DE UNA BASE DE DATOS RELACIONAL

En este punto veremos definiciones aplicadas a la teoría de las bases relacionales muchas de ellas extraídas de las 13 reglas de Codd, son conocidas por las 12 reglas de Codd *CODD'S TWELVE RULES* aunque son 13 porque incluyen una regla 0.

4.1. Generalidades

● En una base de datos relacional, los datos se organizan en **relaciones** *RELATIONSHIP* compuestas por **tuplas** *TUPLES* de **atributos** *ATTRIBUTES*. Si convertimos esta definición a tablas tenemos que los datos se organizan en **tablas** *TABLES* compuestas por **filas** (registros) *ROWS* y **columnas** (campos) *COLUMNS*.

● A cada tabla se le asigna un **nombre único** *UNIQUE NAME*.

● Una tabla tiene 0 o más **filas**, y cada fila contiene la información de un determinado 'sujeto' de la relación.

El equivalente en ficheros convencionales sería:

Relación / tabla --> fichero *FILE*
Tupla/ fila --> registro *RECORD*
Atributo/ columna --> campo *FIELD*

● Las filas en un principio están desordenadas.

● La lista de los atributos dispuestos en un orden específico de izquierda a derecha y que forman la definición de una tabla se denomina **esquema de la tabla** *TABLE SCHEMA*, mientras que los valores concretos de los datos que están almacenados en la tabla se llaman **ocurrencias** *INSTANCE*.

Por ejemplo, tenemos estas dos tablas:

Piezas

Codigo	Denominacion	Precio	Fabricante	Codigo_segun_fab
1	Taburete 3 patas	25	Fab1	T123-34
2	Mesa ovalada	1200	Fab1	M234-87
3	Taburete 3 patas	78	Fab2	T2S-0P

Y

Fabricantes

IdFab	Nombre	Direccion
Fab1	Muebles La Madera	Null
Fab2	Maderas Asociados	Avda. del Sol, nave3
Fab5	Industrias Madereras	Pol. Ind 3 fuentes, nave 23

El esquema de la tabla *Piezas* está compuesto por las columnas (*Codigo*, *Denominación*, *Precio*, *Fabricante*, *Codigo_según_fab*).

Codigo es el código de la pieza, *Denominacion* el nombre de la pieza, *Fabricante* el código del fabricante que nos suministra la pieza y *Código_según_f*ab el código que utiliza ese fabricante para identificar la pieza en su sistema de gestión.

Una ocurrencia de fila de la tabla *Piezas* sería:
*1, Taburete 3 patas, 25, Fab1, T123-3*4.

● En una tabla cada columna tiene un único nombre y éste no se puede utilizar para nombrar otra columna de la misma tabla pero sí de otra tabla.

Por ejemplo en la tabla *Piezas* no se pueden definir dos columnas llamadas *Codigo*, por eso el segundo código lo hemos llamado *Codigo_segun_fab*. Pero en la tabla *Fabricantes* la columna *IdFab* se podía haber llamado *Codigo* sin problema.

● En una tabla no se admiten dos filas con los valores coincidentes en todos sus campos. Esta restricción no se suele cumplir.

Esta regla nos dice que por ejemplo en la tabla *Fabricantes* no pueden haber dos filas con los valores *Fab1, Muebles la Madera, null*.

Realmente sería información redundante, por eso la existencia de esta regla, no obstante en algunos casos muy concretos sí es necesario poder almacenar dos ocurrencias de fila idénticas, por esta razón muchos SGBD no cumplen esta regla.

● Todos los valores de una columna determinada tienen el **mismo tipo de datos** *DATA TYPE*, y éstos están extraídos de un conjunto de valores legales llamado **dominio** *DOMAIN* de la columna. Muchas veces el dominio se corresponderá con un tipo de datos estándar del sistema.

Por ejemplo en la tabla *Piezas* la columna *Codigo* está definida sobre el dominio de los enteros *INTEGER*.

4.2. El valor nulo

● A parte de los valores del dominio, la columna puede contener un valor especial, el **valor nulo** *NULL*. El valor nulo es importante porque representa la ausencia de valor en el campo y no es lo mismo que el valor cero “0” o la cadena vacía o espacios en blanco. De hecho es un valor tan especial que no funciona como los demás valores, por ejemplo no podemos comparar (con el operador de comparación =) un campo con el valor nulo, tenemos que utilizar un operador especial (IS NULL). Incluso se han tenido que redefinir los operadores lógicos para tener en cuenta el valor nulo.

Ej. En la tabla *Fabricantes* el campo *dirección* de la primera fila contiene el valor nulo (*null*) esto significa que este fabricante no tiene dirección (al menos conocida).

4.3. Clave principal - primaria

● Toda tabla debe tener una **clave principal** (**clave primaria**) *PRIMARY KEY* y sólo una.

Una clave primaria es cualquier columna (o combinación de columnas) que permite identificar de forma unívoca cada una de las filas de la tabla. Para que pueda cumplir su cometido, **la clave primaria no puede contener valores nulos ni valores duplicados** *DUPLICATED VALUES* (no podrá haber dos filas con el mismo valor en este campo). Muchas veces en vez de enfocarlo como que no pueden tener valores duplicados decimos que contienen valores únicos *UNIQUE*. *UNIQUE* es el término que se emplea cuando nos queremos referir a este tipo de campos.

Hay SGBD *DBMS* que incluyen el concepto de clave primaria pero no la hacen obligatoria, por lo que en estos sistemas se pueden definir tablas sin clave primaria.

En el ejemplo anterior, en la tabla *Fabricantes* la columna *IdFab* podría ser clave principal, no hay dos fabricantes con el mismo *IdFab* y no hay valores nulos en la columna.

En la tabla *Piezas Codigo* puede ser clave primaria, pero no *Denominación* (hay dos pìezas con la misma denominación).

En una tabla no pueden haber dos claves primarias (por definición), pero sí podemos tener una clave primaria compuesta por dos (o más) columnas. Normalmente se da el caso cuando no hay en la tabla ninguna columna que sirva como clave principal (es decir ninguna columna sin nulos ni valores duplicados) entonces nos vemos obligados a definir una combinación de columnas que sí lo sea. En este caso la combinación de las columnas actúa como si fuese un solo campo.

Por ejemplo, imagina que en la tabla *Piezas* no existiese el campo *Codigo*, y además sabemos que el campo *Codigo_segun_fab* (el *código_segun_fab* es el código que utiliza el fabricante para identificar sus piezas) puede tener valores duplicados porque pueden haber dos piezas diferentes con el mismo *Codigo_segun_fab*, porque dos fabricantes utilizan el mismo codigo para productos diferentes. Entonces en la tabla no tenemos ninguna columna que cumpla las condiciones de una clave primaria. Pero si al campo *Codigo_segun_fab* le añadimos el campo *Fab* (el código del fabricante) la combinación no tiene duplicados porque ningún fabricante asigna a dos priezas diferentes el mismo código, luego la combinación *Fab*+*Codigo_segun_Fab* serviría como clave principal, tendríamos una clave primaria compuesta por dos columnas.

4.4. Clave alternativa

● En una tabla pueden existir más de una columna que permita identificar las filas de la tabla, si queremos utilizar tales columnas como identificadores *IDENTIFIERS* las definiremos como claves **alternativas** *ALTERNATE KEY SURROGATE KEY*. Una clave alternativa tiene las mismas restricciones que una clave primaria (no admite nulos ni duplicados), pero como no podemos definir dos claves primarias, definimos la que se vaya a utilizar más frecuentemente como clave primaria y la otra (u otras) como alternativa.

Por ejemplo en la tabla *Piezas* la clave primaria es el campo *Codigo* ya que no hay ni puede haber dos piezas con el mismo código. Este campo realmente sirve para identificar las filas de la tabla, sabiendo un valor de código (por ejemplo el 2) sabremos que nos referimos a la fila de la mesa ovalada.

En esta misma tabla tenemos una posible clave alternativa, la formada por los campos *Fabricante* y *Codigo_segun_fab* ya que en la tabla *Piezas* es imposible tener dos filas con la misma combinación de valores en estos campos. Luego esta combinación se podría definir como clave alternativa.

Algunos autores hablan de claves secundarias *SECONDARY KEY* para referirse a claves alternativas, mientras que otros autores reservan el término clave secundaria para campos que se utilizan para recuperar la información pero que no cumplen necesariamente con los requisitos de una clave primaria (pueden contener valores duplicados).

4.5. Clave ajena – foránea

● Otro concepto muy importante, fundamental en las bases de datos relacionales, es la **clave ajena** (**externa** o **foránea**) *FOREIGN KEY*.

Una clave ajena es un campo (o combinación de campos) que contiene la referencia a una fila de otra tabla, también puede referirse a la misma tabla. En otras palabras, es un campo que señala a un registro de otra tabla, contiene un valor que identifica un registro de la otra tabla. Son los campos que se utilizan para relacionar las tablas entre sí.

Por ejemplo la pieza *3* es servida por el fabricante *Fab2*, valor que señala al fabricante *Maderas Asociados* en la tabla de fabricantes.

Una tabla puede tener 0, una o varias **claves ajenas**.

● Una **clave ajena puede contener valores duplicados y valores nulos**.

Siguiendo el ejemplo anterior, en la tabla *Piezas* tenemos la clave ajena *Fabricante* ya que en este campo nos guardamos un valor que señala a una fila de la tabla *Fabricantes*, en este campo tenemos el código del fabricante que nos suministra la pieza y este código nos lleva al fabricante correspondiente en la tabla *Fabricantes*. Podemos tener varias piezas suministradas por el mismo fabricante por lo que todas estas piezas tendrían el mismo código de fabricante, la columna admite duplicados. También se podría permitir que una pieza no fuese asignada a ningún fabricante, en este caso en la columna *Fab* de esta pieza habría un valor nulo.

4.6. Reglas de integridad

● El SGBD deberá velar por la integridad de los datos, para ello incluye varias reglas de integridad que se comprobarán de forma automática sin necesidad de la intervención externa de los usuarios o de los programas de aplicación.

- **La integridad de claves**: *KEY INTEGRITY* "Toda tabla debe tener una clave primaria que permite identificar unívocamente los registros que contiene, por lo tanto no puede contener el valor nulo ni valores duplicados".

Si la clave primaria tuviese valores nulos en algunos registros, esos registros no se podrían identificar (si en un grupo tenemos personas que no tienen nombre, no las podemos llamar por su nombre).

Si la clave primaria admitiese valores duplicados tampoco serviría para identificar, si en ese grupo de personas hay tres personas con el mismo nombre (Juan), al llamar a Juan saldrían tres personas y no una única.

En el ejemplo anterior si intentamos insertar una nueva pieza con el código 2, el sistema no nos dejará porque ya hay una pieza con este mismo código en la tabla.

• **La integridad referencial**: *REFERENTIAL INTEGRITY* “En una clave ajena no puede haber un valor no nulo que no exista en la tabla de referencia”. Para que no existan errores de integridad referencial en la base de datos, el sistema comprueba automáticamente que los valores introducidos en las claves ajenas existan en el campo de referencia en la otra tabla; si no existe, no nos dejará insertar el registro.

Volviendo al ejemplo anterior, si intentamos insertar una pieza con un código de fabricante que no existe en la tabla de fabricantes, el sistema no nos dejará.

● A nivel de control sobre los datos, el SGBD debe de proporcionar herramientas para poder definir restricciones de dominio que se comprobarán de forma automática (se comprueba que el valor introducido en una columna pertenece al dominio de la columna, al tipo de datos), y reglas de negocio *BUSINESS RULES*, son reglas específicas sobre los datos, en este tipo de reglas entran las reglas de validación *CHECK RULES* y reglas definidas a nivel superior.

Una regla de validación sería por ejemplo que el precio no pueda ser inferior a 10 euros, y una regla de negocio, que no pueda haber más de 20 fabricantes.

Todas estas reglas se pueden definir cuando definimos la estructura de la base de datos y a partir de este momento no nos tenemos que preocupar de comprobar que se cumplen en todos los formularios/programas que definamos, el sistema las comprobará automáticamente.

4.7. Arquitectura

● Un SGBD relacional sigue la arquitectura de tres niveles ANSI-SPARC *ANSI-SPARC ARCHITECTURE* vista en el tema anterior, en la que tenemos:

En el nivel externo *EXTERNAL LEVEL* las **vistas** *VIEWS*, son consultas que recuperan determinados datos de la base de datos y que actúan como si fuesen tablas con los datos recuperados. La mayoría de los SGBD permiten asignar esas vistas a determinados usuarios lo que nos permite “esconder” información a los usuarios, el usuario sólo verá lo que le permitamos en sus vistas.

En el nivel conceptual *CONCEPTUAL LEVEL* tenemos el esquema conceptual con la definición de todas las tablas, columnas que las componen y relaciones entre ellas sin entrar en el detalle de los tipos de datos de las columnas, las reglas de validación, etc. Es decir es el conjunto del cual extraemos las vistas.

En el nivel interno *INTERNAL LEVEL* tenemos la definición física de la base de datos, es decir la definición de cómo son las tablas y cómo se almacenan los datos así como las reglas de validación y reglas de negocio. Normalmente el administrador de la base de datos *DATABASE ADMINISTRATOR* es el que tiene acceso a esta información, entre otras cosas porque él es el encargado de definir la base de datos, los usuarios *USERS* y programadores *PROGRAMMERS* y quienes tienen acceso a las vistas.

4.8. El lenguaje

● Finalmente tenemos para poder manejar la información almacenada en la base de datos un lenguaje que cumple las reglas de Codd, el lenguaje SQL que introduciremos en próximos temas. Este lenguaje como lo veremos permite manejar la información almacenada de forma rápida y sencilla sin tener que desarrollar complicados programas.

5. Referencias

http://www.databaseanswers.org/codds_rules.htm
http://www.cse.ohio-state.edu/~sgomori/570/coddsrules.html
http://www.computingstudents.com/notes/database_systems/ansi_sparc_3_level_database_architecture.php
http://download.oracle.com/docs/cd/B10500_01/server.920/a96524/c22integ.htm
Only **Introduction to Data integrity** down to **Referential Integrity Rules** (first paragraph).