ESTÁNDARES ANSI SQL

SQL-89

La historia de SQL (Structured Query Language) empieza en 1974 con la definición, (por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación 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. Los experimentos 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, sólo por citar algunos) 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 tenia el nombre de SQL/86. En 1989, ANSI definió el **SQL89**, basado en el anterior pero con una serie de mejoras (definición de claves primarias, integridad de los datos, etc). Una característica importante definida era la posibilidad de utilizarse a través de dos interfaces: interactivamente o dentro de programas de aplicación.

En su primera versión del SQL-89 se tienen tres partes:

- **El lenguaje de definición de datos (LDD).** Contiene todas las instrucciones para definir el esquema de una base de datos, como son: **create**, **alter** y **drop**.

- **El lenguaje de manipulación de datos (LMD).** Contiene las instrucciones de manejo de las tablas como son: **select**, **insert**, **delete** y **update**, y para control de concurrencia como: **commit** y **rollback**.

- **El lenguaje de control de datos (LCD).** Contiene aquellas instrucciones para dar y revocar permisos de acceso a los datos de la base de datos, como son: **grant** y **revoke**.

Todas las instrucciones pueden ir embebidas en programas escritos en otros lenguajes de programación, como: Cobol, Fortran, Pascal t PL/1.

Todas las sentencias SQL comienzan con un verbo, una palabra clave que describe lo que la sentencia hace. CREATE, INSERT, DELETE, COMMIT son verbos típicos. La sentencia continua con una o más cláusulas. Una cláusula puede especificar los datos

sobre los que debe actuar la sentencia, o proporcionar más detalles acerca de lo que la sentencia debe hacer. Todas las cláusulas comienzan también con una palabra clave, tal como WHERE, FROM, INTO y HAVING. Algunas cláusulas son opcionales, otras necesarias. La estructura y contenido específico varían de una cláusula a otra. Muchas cláusulas contienen nombres de tablas o columnas; algunas pueden contener palabras claves adicionales, constantes o expresiones.

SQL-92

SQL-92 fue desarrollado por el comité técnico NCITS H2 sobre bases de datos. Este comité desarrolla estándares para la sintaxis y semántica de los lenguajes de bases de datos. SQL-92 fue diseñado para ser un estándar para los sistemas manejadores de bases de datos relacionales (RDBMS). Esta basado en SQL-89, cuya primera versión se conoce como SQL-86. En 1992 aparece SQL2 o SQL92, la versión hoy en día más difundida ([ISO/IEC 1992] [ANSI 1992] [ISO/IEC 1994]). Con la aparición de la segunda versión del estándar (SQL2) en 1992, prácticamente todos los RDBMS, incluso los no relacionales, incluían soporte a SQL. Hoy en día, SQL se ha convertido en el lenguaje de consulta más utilizado.

SQL (Structured Query Languaje) además de permitirnos consultas en la base de datos, contiene primitivas de definición de tablas, actualización de la base de datos, definición de vistas otorgamientos de privilegios, etc. A continuación se mostrarán aspectos del estándar ANSI de 1992, conocido como SQL-92.

Definición de tablas y esquemas

Definición de Esquemas

La definición de un esquema es simple. Sólo se necesita identificar el comienzo de la definición con una instrucción CREATE SCHEMA y una cláusula adicional AUTHORIZATION y a continuación definir cada dominio, tabla, vista y demás en el esquema

Por ejemplo:

```
CREATE SCHEMA EMPRESA_CL
        AUTHORIZATION DUEÑO
        definición de dominios
        definición de tablas
        definición de vistas
         etc.
```

El dueño del esquema, o propietario del esquema puede otorgar privilegios de acceso y actualización de la base de datos definida en el esquema a otros usuarios del sistema.

Tipos de datos y dominios

Un dominio es un conjunto del cual toma sus valores una columna de una relación. Según este concepto, los tipos de datos predefinidos son dominios. Adicionalmente SQL-92 permite la definición de dominios por parte de los usuarios.

Numéricos exactos:

Integer	enteros
Small Integer	enteros pequeños
Numeric(p, e)	p: precisión: total de números o dígitos en el número e: escala: cuantos números están a la derecha del punto decimal
Decimal(p,e)	p:precisión e: escala

Numéricos aproximados:

Real	
Double precision	doble precisión
float	flotante

Estos tipos de datos se utilizan normalmente para cálculos científicos y de ingeniería.

Cadenas de caracteres:

Character(n)	carácter
Character varying(n)	carácter variable

Los campos de character siempre almacenan *n* caracteres, aún cuando tengan que rellenar con blancos a la derecha para completar la longitud *n*. Los campos character varying sólo almacenan el número real de caracteres que se introdujeron (hasta un máximo de *n*).

Cadenas de bits:

Bit(n)
Bit varying(n)

Estos campos se usan para banderas u otras máscaras de bits para el control.

Fechas y horas:

Date	fecha
Time	hora
Timestamp	sello de tiempo
Time con tiempo zonal	
Timestamp con tiempo zonal	

El tipo Date se da en el orden año, mes, día con cuatro dígitos para el año. El Time se da en horas (0 a 23), minutos, segundos y décimas de segundos. El Timestamp es la fecha más la hora.

Intervalos:

Year-month	año-mes
Day-time	día-hora

Un intervalo es la diferencia entre dos fechas (año-mes) o entre dos horas (día-hora).

Definición de dominios:

Los tipos de datos con restricciones (constrains) y valores por defecto (default values) se pueden combinar en la definición de dominios. Una definición de dominio es un tipo de datos especializado que puede estar definido dentro de un esquema y utilizado en la definición de columnas.

Por ejemplo:

```
CREATE DOMAIN IDENTIFICADOR NUMERIC(4) DEFAULT 0
        CHECK (VALUE IS NOT NULL)
```

Esta definición dice que un dominio llamado IDENTIFICADOR tiene las siguientes propiedades:

1. Su tipo de datos es numérico de cuatro dígitos.
2. Su valor por defecto es 0.
3. Nunca puede ser nulo.

Definición de Tablas

Las tablas se definen en tres pasos:

1. Dar el nombre de la tabla.
2. Definir cada columna, posiblemente incluyendo restricciones de columna.
3. Definir las restricciones de la tabla.

Siguiendo el ejemplo de EMPRESA_CL y del dominio IDENTIFICADOR:

```
CREATE SCHEMA EMPRESA_CL
        AUTHORIZATION DUEÑO

CREATE DOMAIN IDENTIFICADOR NUMERIC(4) DEFAULT 0
        CHECK (VALUE IS NOT NULL)

CREATE TABLE TRABAJADOR (
    TRA_ID       IDENTIFICADOR   PRIMARY KEY,
    TRA_NOMBRE   CHARACTER(12),
    TRA_TARIFA_HR  NUMERIC(5,2),
    TRA_OFICIO   CHARACTER(8),
    TRA_SUP      NUMERIC(4),
    FOREIGN KEY ID_SUPV REFERENCES TRABAJADOR
      ON DELETE SET NULL
)

CREATE TABLE ASIGNACION (
      ASG_ID_TRABAJADOR   IDENTIFICADOR,
      ASG_ID_EDIFICIO     IDENTIFICADOR,
      ASG_FECHA_INICIO    DATE,
      ASG_NUM_DIAS        INTERVAL DAY (3),
      PRIMARY KEY (ASG_ID_TRABAJADOR, ASG_ID_EDIFICIO),
      FOREIGN KEY ASG_ID_TRABAJADOR REFERENCES TRABAJADOR
          ON DELETE CASCADE,
      FOREIGN KEY ASG_ID_EDIFICIO REFERENCES EDIFICIO
          ON DELETE CASCADE
)
```

```
CREATE TABLE EDIFICIO (
        EDI_ID          IDENTIFICADOR    PRIMARY KEY,
        EDI_DIRECCION     CHARACTER(12),
        EDI_TIPO          CHARACTER(9) DEFAULT 'Oficina'
            CHECK (TIPO IN ('Oficina', 'Almacén', 'Comercio',
                'Residencia')),
        EDI_NIVEL_CALIDAD  NUMERIC(1),
        EDI_CALIDAD       NUMERIC(1) DEFAULT 1
            CHECK (CATEGORIA > 0 AND CATEGORIA < 4)
  )
```

Después del CREATE SCHEMA y de la definición de dominios (CREATE DOMAIN) van las instrucciones de creación de tablas. La instrucción CREATE TABLE identifica el nombre de la tabla, que debe ser única dentro del esquema. Después del CREATE TABLE van encerradas entre paréntesis y separadas por coma las instrucciones de definición de columnas y restricciones sobre la tabla.

Las definiciones de columnas de la tabla (que son los atributos del modelo relacional) están compuestas por:

NombreColumna TipoDeDatos [ValorPorDefecto] [RestriccionesEspecíficas]

El [ValorPorDefecto] y las [RestriccionesEspecíficas] pueden no especificarse, por ejemplo:

```
        EDI_DIRECCION     CHARACTER(12),
```

especificar solamente restricciones específicas:

```
        EDI_ID          IDENTIFICADOR    PRIMARY KEY,
```

especificar solamente el valor por defecto:

```
        EDI_TIPO          CHARACTER(9) DEFAULT 'Oficina',
```

o especificar valor por defecto y restricciones:

```
        EDI_CALIDAD       NUMERIC(1) DEFAULT 1
                          CHECK (CATEGORIA > 0 AND CATEGORIA < 4)
```

Las restricciones de la tabla son por ejemplo:

```
        FOREIGN KEY ID_SUPV REFERENCES TRABAJADOR
                ON DELETE SET NULL
        PRIMARY KEY (ASG_ID_TRABAJADOR, ASG_ID_EDIFICIO),
        FOREIGN KEY ASG_ID_TRABAJADOR REFERENCES TRABAJADOR
                ON DELETE CASCADE,
        FOREIGN KEY ASG_ID_EDIFICIO REFERENCES EDIFICIO
                ON DELETE CASCADE
```

Analicemos la definición de las llaves:

```
        PRIMARY      KEY      (ASG_ID_TRABAJADOR,      ASG_ID_EDIFICIO),
```

significa que ASG_ID_TRABAJADOR y ASG_IS_EDIFICIO son las llaves de la tabla, por lo tanto los valores combinados de estas dos columnas deben ser únicos para la tabla. Por otro lado dado que son FOREIGN KEY sabemos que son llaves externas.

```
        FOREIGN KEY ASG_ID_TRABAJADOR REFERENCES TRABAJADOR
                ON DELETE CASCADE,
```

es una clave externa recursiva (referencia a otra fila de su propia relación).

```
        FOREIGN KEY ASG_ID_EDIFICIO REFERENCES EDIFICIO
                ON DELETE CASCADE
```

Es una clave externa normal (referencia a otra relación).

Las palabras ON DELETE indican que hacer en caso de borrar al referido. La cláusula ON DELETE SET NULL le dice al sistema que si se borra la tupla a la que apunta la clave externa entonces el valor de ésta se debe poner a cero, esto sirve para mantener la integridad de referencias (si borro al referido, y quedan referencias apuntando a una tupla borrada tenemos una falla en la integridad de referencia). La cláusula ON DELETE CASCADE significa que si se borra la tupla referida en la relación, se realiza un borrado en "cascada" de todas las tuplas que hacían referencia a ésta.

Similar a ON DELETE tenemos a ON UPDATE y ambas cláusulas tienen las opciones siguientes:

- CASCADE
- SET NULL
- SET DEFAULT

Por último se debe decir que a diferencia de las relaciones en el modelo relaciona, no se requiere que cada tabla SQL tenga una clave primaria. En otras palabras, si no se indica ninguna clave, entonces dos filas de la tabla pueden tener valores idénticos. Lo malo es que una tabla que no tenga clave primaria no puede ser referida mediante clave externa desde otra tabla.

Adicionalmente SQL-92 define instrucciones para cambiar las definiciones de las tablas (ALTER TABLE) o para borrar las tablas (DROP TABLE). Para borrar un esquema completo se usa DROP SCHEMA, pero como borrar un esquema es una cosa seria, se tiene que especificar que opción de borrado se usa:

DROP SCHEMA NombreEsquema CASCADE: significa eliminar el esquema con ese nombre al igual que todas las tablas, datos y otros esquemas que aún existan.
DROP SCHEMA NombreEsquema RESTRICT: significa eliminar el esquema sólo si todos los restantes objetos del esquema ya han sido borrados.

Manipulación de datos

Consultas simples

Consultas simples: Una consulta que involucra una sola tabla de la base de datos.

Ejemplo: ¿Quiénes son los fontaneros?

```
SELECT TRA_NOMBRE
        FROM TRABAJADOR
 WHERE TRA_OFICIO = 'Fontanero'
```

Aquí se muestran las tres cláusulas más usadas en SQL: la cláusula SELECT, la cláusula FROM y la cláusula WHERE. Una buena regla es escribir cada cláusula en una línea aparte y con sangrías, aunque se puede escribir todo en la misma línea.

Cláusula SELECT: Señala las columnas que se desean en la consulta (equivalente a la proyección del álgebra relacional).

Cláusula FROM: Lista las tablas que son referidas por la consulta.

Cláusula WHERE: Nos da la condición para seleccionar las filas de las tablas indicadas.

La sentencia SQL anterior se procesa por el sistema en el orden FROM, WHERE, SELECT.

Otro ejemplo: ¿Cuál es la tarifa semanal de cada electricista?, además entregue la salida ordenada alfabéticamente.

```
SELECT TRA_NOMBRE, 'Tarifa semanal = ', 48 * TRA_TARIFA_HR
FROM TRABAJADOR
WHERE TIPO = 'ELECTRICISTA'
ORDER BY TRA_NOMBRE
```

Debe notarse la inclusión de la cadena de caracteres 'Tarifa semanal = ' al cálculo de la consulta. También nótese que con la cláusula ORDER BY podemos ordenar la salida, si se quisiera orden descendente, se utilizaría "DESC".

Consulta: ¿Quiénes tienen una tarifa por hora entre $7000 y $9000, entregue toda la información de la tabla?

```
SELECT *
FROM TRABAJADOR
WHERE TRA_TARIFA_HR >= 7000 AND TRA_TARIFA_HR <= 9000
```

Se pueden usar los siguientes operadores de comparación: =, <> (distinto), <, >, <=, >=.

Se pueden usar los siguientes conectores booleanos: AND, OR, NOT.

Consulta: Indique los fontaneros, albañiles y electricistas.

```
SELECT *
FROM TRABAJADOR
WHERE TRA_OFICIO IN ('Fontaneros', 'Albañiles', 'Electricistas')
```

Consulta: Encontrar todos los trabajadores cuyos oficios comiencen con "Elec". Para esto necesitamos usar caracteres comodines. Es decir, símbolos especiales que valen por cualquier cadena de caracteres.

```
SELECT *
FROM TRABAJADOR
WHERE TRA_OFICIO LIKE 'Elec%'
```

SQL tiene dos caracteres comodines, el % que vale por cero o cualquier cantidad de caracteres, y _ que vale por exactamente un caracter cualquiera. El operador LIKE se usa para comparar variables de caracteres literales cuando se utilizan comodines.

Consulta: Encuentre todas las asignaciones que comiencen en las dos próximas semanas.

```
SELECT *
FROM ASIGNACION
WHERE ASG_FECHA_INICIO BETWEEN CURRENT_DATE AND CURRENT_DATE +
INTERVAL '14' DAY
```

Aquí se introduce el operador BETWEEN, que devuelve VERDADERO si el valor está dentro del intervalo [CURRENT_DAY, CURRENT_DATE + INTERVAL '14' DAY] y FALSO si no. CURRENT_DAY (fecha actual) es una función que siempre devuelve la fecha de hoy.

Consultas multi-tablas

Consulta: ¿Cuáles son los oficios de los trabajadores asignados al edificio 435?

```
SELECT TRA_OFICIO
FROM TRABAJADORES, ASIGNACION
WHERE TRA_ID = ASG_ID_TRABAJADOR
AND ASG_ID_EDIFICIO = 435
```

En este caso, la cláusula FROM calcula el producto cartesiano de las tablas indicadas y luego con la cláusula WHERE filtramos las filas interesantes (aquellas en que TRA_ID es igual a ASG_ID_TRABAJADOR) y luego las que nos interesan (las asignaciones al edificio 435), y por último SELECT toma la columna que nos interesa. Ojo que cuando hay problemas de que se repite un nombre de columna le ponemos como prefijo el nombre de la tabla seguida por un punto, ejemplo: TRABAJADORES.TRA_ID, o ASIGNACION.ASG_ID_EDIFICIO.

Consulta: Indicar los trabajadores con los nombres de sus supervisores

```
SELECT A.TRA_NOMBRE, B.TRA_NOMBRE
FROM TRABAJADORES A, TRABAJADORES B
WHERE A.TRA_SUP = B.TRA_ID
```

En este caso, para resolver la consulta necesitamos dos copias de la tabla, para ello usamos *alias* (un nombre alternativo que se le da a una relación) de las tablas A y B para referirnos a cada copia.

Subconsultas

Subconsulta: Una consulta dentro de una consulta.

Consulta: ¿Cuáles son los oficios de los trabajadores asignados al edificio 435?

```
SELECT TRA_OFICIO
FROM TRABAJADOR
WHERE TRA_ID IN (
  SELECT ASG_ID_TRABAJADOR
  FROM ASIGNACION
  WHERE ASG_ID_EDIFICIO = 435
)
```

En el ejemplo la subconsulta es:

```
(
  SELECT ASG_ID_TRABAJADOR
```

```
  FROM ASIGNACION
  WHERE ASG_ID_EDIFICIO = 435
)
```

A veces es posible realizar una sola consulta, sin subconsulta, pero más compleja:

```
SELECT OFICIO
FROM TRABAJADORES, ASIGNACIONES
WHERE ASG_ID_EDIFICIO = 435
  AND TRA_ID = ASG_ID_TRABAJADOR
```

EXISTS y NOT EXISTS

Operador EXISTS: Evalúa verdadero si el conjunto resultante es no vacío.

Operador NOT EXISTS: Evalúa verdadero si el conjunto resultante es vacío.

Consulta: ¿Quienes son los trabajadores que no están asignados al edificio 435?

```
SELECT TRA_ID, TRA_NOMBRE
FROM TRABAJADOR
WHERE NO EXISTS (
  SELECT *
  FROM ASIGNACION
  WHERE ASG_ID_TRABAJADOR = TRA_ID
    AND ASG_ID_EDIFICIO = 435
)
```

Los operadores EXISTS y NOT EXISTS siempre preceden a una subconsulta.

Además esta subconsulta tiene apellido, es una subconsulta correlacionada, es decir, es una subconsulta cuyos resultados dependen de la fila que se está examinando por una consulta más externa.

Funciones integradas

Consulta: ¿Cuáles son la tarifa por hora mayor y menor?

```
SELECT MAX(TRA_TARIFA_HR), MIN(TRA_TARIFA_HR)
FROM TRABAJADOR
```

Consulta: ¿Cuál es el promedio de días que los trabajadores están asignados al edificio 435?

```
SELECT AVG(ASG_NUM_DIAS)
FROM ASIGNACION
WHERE ASG_ID_EDIFICIO = 435
```

Consulta: ¿Cuál es el número total de días asignados a fontanería en el edificio 312?

```
SELECT SUM(ASG_NUM_DIAS)
FROM ASIGNACION, TRABAJADOR
WHERE TRA_ID = ASG_ID_TRABAJADOR
  AND TRA_OFICIO = 'Fontanero'
  AND ASG_ID_EDIFICIO = 312
```

Consulta: ¿Cuántos tipos de oficios diferentes hay?

```
SELECT COUNT(DISTINC TRA_OFICIO)
FROM TRABAJADOR
```

La palabra clave DISTINCT se usa para que el sistema no cuente el mismo tipo de oficio mas de una vez.

Como muestran todos estos ejemplos, si una función integrada aparece en una cláusula SELECT, entonces nada más que funciones integradas pueden aparecer en dicha cláusula SELECT. La una excepción ocurre en combinación con la cláusula GROUP BY.

GROUP BY y HAVING

Cláusula GROUP BY: Indica cuáles filas deben agruparse sobre un valor común de las columna(s) especificada(s)

Cláusula HAVING: Una cláusula que impone condiciones a los grupos.

A diferencia de ORDER BY que se ejecutan sólo para ordenar la salida, GROUP BY y HAVING permiten generar particiones (grupos de datos) para realizar operaciones sobre las particiones.

Consulta: Para cada supervisor, ¿Cuál es la tarifa por horas más alta que se le paga a un trabajador que informe a este supervisor?

```
SELECT TRA_SUP, MAX(TRA_TARIFA_HR)
FROM TRABAJADOR
GROUP BY TRA_SUP
```

Consulta: Para cada supervisor que dirige a más de un trabajador, ¿cuál es la tarifa por horas más alta que se le paga a un trabajador que informe a dicho supervisor?

```
SELECT TRA_SUP, MAX(TRA_TARIFA_HR)
FROM TRABAJADOR
GROUP BY TRA_SUP
HAVING COUNT(*) > 1
```

Noten que la cláusula WHERE aplica a las filas, en cambio la cláusula HAVING a grupos !!, aunque pueden trabajar juntos WHERE y HAVING.

Consulta: Para cada supervisor que dirige a más de un trabajador, ¿cuál es la tarifa por horas más alta que se le paga a un electricista que informe a dicho supervisor?

```
SELECT TRA_SUP, MAX(TRA_TARIFA_HR)
FROM TRABAJADOR
WHERE TRA_OFICIO = 'Electricista'
GROUP BY TRA_SUP
HAVING COUNT(*) > 1
```

Operaciones de modificación de la base de datos

NSERT: Operación que causa que se añadan filas a una relación.

UPDATE: Operación que cambia los valores de las columnas en las filas.

DELETE: Operación que quita filas de una relación.

Inserción:

```
INSERT INTO ASIGNACION (ASG_ID_TRABAJADOR, ASG_ID_EDIFICIO,
ASG_FECHA_INICIO)
VALUES (1284, 485, '2002-05-13')
```

Actualización:

```
UPDATE TRABAJADOR
SET TRA_TARIFA_HR = 1.05 * TRA_TARIFA_HR
WHERE ID_SUPV = 1520
Borrado:
DELETE FROM TRABAJADOR
WHERE ID_SUPV 1520
```

Definición de vistas

Una vista es una porción restringida de la base de datos, y se obtiene de una tabla que contiene información básica o real. Por ejemplo si quisieramos crear una vista de la tabla TRABAJADOR pero que no muestre su tarifa por hora:

```
CREATE VIEW TRABAJADOR_B
AS SELECT TRA_ID, TRA_NOMBRE., TRA_OFICIO, TRA_SUP
  FROM TRABAJADOR
```

En general para definir una vista se utiliza:

```
CREATE VIEW NombreVista
AS ConsultaSQL
```

SQL 3

Antecedentes

El lenguaje estándar llamado SQL3, prometió ser un aumento de la segunda generación de SQL (comúnmente conocido como SQL92, debido al año de su publicación), SQL3 fue originalmente planeado para su uso en el año 1996, pero tardó 7 años en desarrollarse en vez de los tres o cuatro que se pensaba iba a tardar.

SQL3 está caracterizado como "SQL orientado a objetos" y es la base de algunos sistemas de manejo de bases de datos orientadas a objetos (incluyendo ORACLE, Informix Universal Server, IBM's DB Universal Database y Cloudscape, además de otros).

SQL:1999 envuelve características adicionales que se consideran herencia de los SQL relacionales, así como también una reestructuración de los documentos de los estándar con vista a una mayor progresión hacia normas más efectivas.

Proceso de desarrollo de normas:

Las dos organizaciones que se involucraron en la estandarización de SQL, y por lo tanto en el desarrollo de SQL:1999 son ANSI e ISO. Más específicamente, la comunidad internacional de trabajos mediante ISO/IEC JTC1 (Joint Technical Committee 1), un comité formado por la organización internacional de estandarización junto con la comisión internacional electrotécnica. La responsabilidad de las JTC1 es desarrollar y mantener la información relativa a la tecnología. Dentro de JTC1, el subcomité SC32, cuya función era el intercambio y gestión de datos se formó para la estandarización de normas relativas a varias bases de datos y metadatos que habían sido desarrollados por otras organizaciones (tal como el ahora disuelto SC21). SC32, es a la vez, un número de grupos de trabajo que actualmente realizan los trabajos técnicos-WG3(lenguajes de bases de datos) es responsable de las normas de SQL, mientras WG4 está desarrollando el SQL/MM (SQL multimedia, un departamento de normas que especifiquen las bibliotecas de tipos usando facilidades de SQL orientado a objetos).

En los Estados Unidos, IT estándares son manejados por la institución de acreditación y desarrollo de normas nacionales estadounidenses, el comité NCITS(Comité nacional para la estandarización de tecnología de la información, anteriormente conocido como X3). NCITS comité técnico H2 (anteriormente X3H2) es responsable de varios estándares relativos a la gestión de datos, incluyendo SQL y SQL/MM.

Cuando la primera generación de SQL fue desarrollada (SQL-86 y su aumento menor SQL-89), casi todo el proceso se hizo en Estados Unidos por X3H2 y otras naciones participaron en su mayor parte en el modo de revisar y criticar el trabajo propuesto por ANSI.

En el momento que SQL-89 fue publicado, la comunidad internacional hizo por escrito propuestas para la especificación que al final llegó a ser SQL-92; que no ha cambiado mientras SQL:1999 está siendo desarrollado.

Las primeras versiones de la norma son conocidas como SQL-86 (o SQL-87, porque la versión no fue publicada hasta 1987), SQL-89 y SQL-92, mientas que la versión actual debe llegar a ser conocida como SQL:1999. ¿Por qué no SQL-99?, Simplemente porque se debe pensar en el nombre de la próxima generación y SQL-02 puede ser confundido con SQL2 (que era el proyecto bajo el cual fue desarrollado SQL-92). En otras palabras, se espera que desde el año 2000 no se produzcan problemas con los nombres de SQL.

Contenidos de SQL:1999

A continuación se van a describir los aspectos nuevos de esta generación en desarrollo de SQL. Los aspectos pueden ser divididos en "aspectos relacionales" y "aspectos relacionados con objetos".

1.Aspectos relacionales

Es más adecuado llamar a esta categoría como "aspectos que relacionan el papel de SQL en el modelado de datos". Estos aspectos no están estrictamente limitados al modelo relacional, pero no están relacionados con la orientación a objetos.

Estos aspectos se dividen en cinco grupos: nuevos tipos de datos, nuevos predicados, semántica mejorada, seguridad adicional, la base de datos activa. Hablaremos de cada uno de los grupos por separado.

1.1.Nuevos tipos de datos

SQL:1999 tiene cuatro nuevos tipos de datos (aunque alguno de ellos tiene variantes identificables). El primero de estos tipos es LARGE OBJECT(objeto grande) o LOB. Las variantes de este tipo son las siguientes:

CHARÁCTER LARGE OBJECT(CLOB)
BINARY LARGE OBJECT(BLOB)

CLOB funciona como una cadena de caracteres, pero tiene restricciones que impiden su uso como PRIMARY KEY o predicados UNIQUE, FOREIGN KEY, y en comparaciones a excepción de las de igualdad o desigualdad.
BLOB tiene restricciones similares. (Por implicación, LOB no puede ser usado en las cláusulas GROUP BY u ORDER BY). Las aplicaciones típicamente no podrán transferir el valor entero de un LOB hacia una base de datos (después del almacenado inicial); pero podría manipular valores LOB mediante un tipo de cliente especial que lo soporte denominado "LOB locator".

En SQL:1999, un locator es un único valor binario que actúa como sustituto para un valor que está dentro de la base de datos; los locators pueden ser usados en operaciones (tales como SUBSTRING)

Otro tipo de dato nuevo es el BOLEAN, que permite a SQL registrar valores de verdad, falso y desconocido. Complejas combinaciones de predicados pueden ser ahora expresadas de una manera más sencilla que antes.

Ejemplo:

```
WHERE COL1>COL2 AND
                    COL3=COL4 OR
                    UNIQUE(COL6) IS NOT FALSE;
```

SQL:1999 también tiene dos nuevos tipos compuestos: ARRAY y ROW. El tipo ARRAY permite almacenar una colección de valores directamente en una columna de una tabla.

Ejemplo:

```
WEEKDAYS    VARCHAR(10)    ARRAY(7)
```

Con esto se podrían almacenar los nombres de los días de la semana en una columna en la base de datos.

¿Esto significa que SQL:1999 permite bases de datos que no satisfagan la 1ª forma normal?.

Desde luego, lo hace en el sentido de que permite los grupos repetitivos, que la primera forma normal prohíbe. (Sin embargo, algunos sostienen que los tipos ARRAY de SQL:1999 meramente permiten almacenar información que puede ser descompuesta, muchas como la función SUBSTRING puede descomponer strings de caracteres y por lo tanto no infringen la 1ª forma normal).

El tipo ROW en SQL:1999 permite el almacenamiento estructurado de datos en columnas únicas de la base de datos.

Ejemplo:
```
CREATE TABLE employee(
EMP_ID INTEGER,
NAME ROW(
GIVEN VARCHAR(30)
FAMILY VARCHAR(30)),
ADDRESS ROW(
STREET VARCHAR(50),
CITY VARCHAR(30),
STATE CHAR(2)),
SALARY REAL)
SELECT E.NAME.FAMILY
FROM employee E;
```

Mientras se podría argumentar que esto también infringe la primera forma normal, la mayoría de observadores reconocen que simplemente es otro tipo de datos descompuesto.

SQL:1999 añade también facilidades para las relaciones entre tipos de datos distintos.

Reconociendo que sería inverosímil comparar directamente el tamaño de zapato de un empleado con su "IQ", el lenguaje permite declarar SHOE_SIZE e IQ como entero, pero prohíbe que se mezclen en expresiones. Así, una expresión como:

```
WHERE MY_SHOESIZE > MY_IQ
```

Se reconoce como un error de sintaxis. Cada uno de estos tipos puede ser representado como un entero, pero no permite su mezcla en expresiones así como tampoco multiplicar por otro entero:

```
SET MY_IQ = MY_IQ * 2
```

Además de estos tipos, SQL:1999 tiene también definidos tipos definidos por el usuario, pero esto se analizará dentro de la orientación a objetos.

1.2.Nuevos predicados:

SQL:1999 tiene tres nuevos predicados, uno de los cuales se analizará conjuntamente con la orientación a objetos. Los otros dos son el predicado SIMILAR y el predicado DISTINTO. Desde la primera versión del SQL estándar, las cadenas de caracteres están limitadas a simples comparaciones (como =, > ó <>) y las rudimentarias capacidades del predicado like:

```
WHERE NAME LIKE '%SMIT_'
```

Con esto diremos que hay cero o más caracteres que preceden a los cuatro caracteres 'SMIT' y exactamente uno después de ellos (tal como SMITH o HAMMERSMITS). Reconociendo que las aplicaciones requieren capacidades mas sofisticadas, SQL:1999 introdujo el predicado SIMILAR que ofrece la posibilidad de equiparar modelos.

Ejemplo:
WHERE NAME SIMILAR TO
'(SQL-(86|89|92|99)) | (SQL(1|2|3))'

(que equipararía los diversos nombres dados al SQL a lo largo de los años). Es un poco desafortunado que la sintaxis de las expresiones regulares usadas en el predicado SIMILAR no saquen partido de las expresiones regulares de UNIX, pero algunos caracteres de UNIX son usados para otros propósitos en SQL.

El otro nuevo predicado, DISTINCT, es muy similar al predicado UNIQUE; la diferencia importante es que dos valores nulos eran considerados no iguales el uno del otro y de este modo podrían satisfacer el predicado UNIQUE, pero no en todas las aplicaciones se daba este caso. El predicado DISTINCT considera que dos valores nulos no pueden ser distintos el uno del otro (o son iguales o distintos) así que dos valores nulos podrían hacer que un predicado DISTINCT no sea satisfactorio.

1.3.Nueva semántica en SQL:1999

Es difícil saber exactamente fijar un límite a la hora de hablar de la nueva semántica en SQL:1999, pero se dará una selección de lo que se cree más importante.

Una de las demandas de los escritores de aplicaciones era ampliar las clases de vistas. Muchos entornos usan vistas pesadas como mecanismo de seguridad y/o como simplificación de una aplicación vista de la base de datos. Como siempre, si la mayoría de vistas no son actualizables, luego estas aplicaciones frecuentemente tienen que escapar de los mecanismos de vista y confiar directamente en la modificación de las tablas subyacentes; esto es una situación poco satisfactoria.

SQL:1999 ha incrementado significativamente el número de vistas que pueden ser modificadas directamente, usando solo lo que se facilitó en el estándar. Esto depende en gran medida de las dependencias funcionales para ir determinando que vistas pueden ser modificadas, y como hacer estos cambios en la tabla de la base de datos para efectuar estas modificaciones.

Otra deficiencia criticada en gran medida de SQL era la imposibilidad de construir aplicaciones recursivas. SQL:1999 está provisto para hacer llamadas recursivas, cosa que satisface lo dicho con anterioridad. Escribiremos la pregunta en base a la cual deseamos hacer la recursión y daremos un nombre. Luego usaremos el nombre asociando la pregunta a una expresión:

WIT RECURSIVE
Q1 AS SELECT .....FROM.....WHERE.....,
Q2 AS SELECT…….FROM…….WHERE…..
SELECT…….FROM Q1,Q2 WHERE……

Se ha mencionado ya a los locators como un valor del cliente que puede representar un valor LOB almacenado en el servidor. Los locator pueden ser usados en algunos casos para representar los valores de un ARRAY, aceptando el hecho de que (como LOB) los array pueden ser frecuentemente demasiado grandes para ser pasado convenientemente entre una aplicación y la base de datos. Los locator pueden ser usados también para representar tipos de valores de datos indefinidos (luego lo discutiremos) que también tienen la potencialidad de ser grandes y abultados.

Finalmente, SQL:1999 ha agregado la noción de savepoints, ampliamente implementado en sus productos. Un savepoint es como una pequeña substracción en la que una aplicación puede deshacer las acciones realizadas después de comenzar el savepoint sin deshace todas las acciones de una transacción entera.

1.4.Mejoras de Seguridad

Las nuevas facilidades de seguridad en SQL:1999 tienen un papel muy importante. Los privilegios pueden ser otorgados según un rol y este a su vez puede otorgar privilegios individuales para otros roles. Esta estructura anidada mejora el manejo de la seguridad en el ambiente de una base de datos. Los roles han estado siendo implementados por los productos SQL desde hace varios años (aunque ocasionalmente bajo nombres diferentes).

1.5.Base de datos activa

SQL:1999 reconoce la noción de base de datos activa. Esto es facilitado por lo que se conoce como triggers (disparadores). Un trigger, como se sabe, es una facilidad que permite a los diseñadores de bases de datos realizar operaciones seguras siempre que una aplicación realice operaciones en tablas particulares. Por ejemplo, los triggers podrían emplearse para registrar todas las operaciones que cambien salarios en la tabla empleado.

```
CREATE TRIGGER log_salupdate

BEFORE UPDATE OF salary
ON employees

REFERRENCING OLD ROW as oldrow
                              NEW ROW as newrow
FOR EACH ROW
INSERT INTO log_table

VALUES(CURRENT_USER,
oldrow.salary,
newrow.salary)
```

Los triggers pueden ser usados para muchos propositos, no solo para registrar. Por ejemplo, tu puedes escribir triggers que guarden un balance de presupuesto para saber si puedes contratar a nuevos empleados o no.

2.Orientación a objetos

Además de las características discutidas hasta ahora, SQL:1999 se caracteriza porque fue desarrollado principalmente para manejar objetos. Algunas de las características que están dentro de esta categoría fueron definidas en el estándar SQL/PSM publicado en 1996 específicamente para llamadas a funciones y procedimientos desde SQL. SQL:1999 mejora esta capacidad que llamó SQL-invoked routines, para añadir una tercera clase de rutina conocida como método, que luego veremos.

2.1.Tipos de estructuras definidas por el usuario

La mayor facilidad en SQL:1999 que soportan la orientación a objetos son los tipos estructurados definidos por el usuario; Los tipos estructurados tienen un número de características, las más importantes son:

Pueden ser definidos para tener uno o más atributos, cada uno de ellos pueden ser algún tipo de SQL, incluyendo tipos empotrados como INTEGER, tipos de colección como ARRAY, u otro tipo de estructuras.
Todos los aspectos de su comportamiento son provistos mediante métodos,funciones y procedimientos.
Sus atributos son encapsulados mediante el uso del sistema generador observador y mutador de funciones (funciones get y set) que provee el único acceso a susvalores. Sin embargo, este sistema no puede ser sobrecargado; todas las otras funciones y métodos pueden ser sobrecargados.
Las comparaciones de sus valores son únicamente realizadas mediante funciones definidas por el usuario.
Ellos pueden participar en jerarquías de tipo, en las cuales más tipos especializados (subtipos) tienen todos sus atributos y usan todas las rutinas asociadas con más tipos generalizados (supertipos), pero pueden agregar nuevos atributos y rutinas.

Veamos un ejemplo de la definición de un tipo estructurado:

```
CREATE TYPE emp_type
UNDER person_type

AS (EMP_ID INTEGER,
SALARY REAL)

INSTANTIABLE
NOT FINAL
REF(EMP_ID)
INSTANCE METHOD
GIVE_RAISE
(ABS_OR_PCT BOOLEAN,
AMOUNT REAL)
RETURNS REAL
```

Este nuevo tipo es un subtipo de otro tipo estructurado que podría ser usado para describir personas en general, incluyendo atributos comunes parecidos como name y address; los atributos del nuevo emp_type incluyen cosas que personas ancianas no tienen, como un ID de empleado y un salario. Nosotros hemos declarado este tipo para ser instanciable y permitir que tenga subtipos definidos (NOT FINAL). En suma, nosotros diremos que algunas referencias a este tipo son derivadas del valor del ID del empleado.
Finalmente definimos un método que puede ser aplicado a instancias de este tipo. SQL:1999, después de una extensiva coquetería con la herencia múltiple (en la cual los subtipos se les permitió tener más de un supertipo inmediato), ahora tiene un modelo de tipo estrechamente relacionado con la herencia simple de Java. Los definidores de tipo se permiten especificar que un tipo determinado es instanciable (en otras palabras, valores de qué tipo específico pueden ser creados) o no instanciable (análogo a los tipos abstractos en otros lenguajes de programación). Y naturalmente, algún lugar –tal como una columna- donde un valor de algún tipo estructurados es permitido, un valor de alguno de estos subtipos pueden aparecer; esto provee la clase de substitutibilidad de la cual los programas orientados a objetos dependen.

Por la manera, algunos lenguaje de programación orientada a objetos, tal como C++ permite definidores de tipo para saber el grado de encapsulamiento de sus tipos: un nivel PUBLIC de encapsulamiento aplicado a un atributo significa que cualquier usuario del tipo puede acceder al atributo, PRIVATE significa que el atributo no puede ser usado por el código de otro método, y PROTECTED significa que solo el método del tipo y métodos del subtipo del tipo pueden acceder al atributo.

SQL:1999 no tiene este mecanismo, aunque se intentó; nosotros anticipamos que esto puede ser propuesto para una futura revisión del estándar.

2.2.Funciones v/s métodos

SQL:1999 hace una importante distinción entre las típicas llamadas a funciones y las llamada a métodos. En resumen, un método es una función con varias restricciones y aumentos.

Dejaremos resumidas las diferencias entre los dos tipos de rutina:
Los métodos están estrechamente limitados a un simple tipo definido por el usuario.
Las funciones pueden ser polimórficas (sobrecargadas), pero una función específica es elegida por compilar en tiempo para examinar las declaraciones de tipos de cada argumento de una invocación a una función y eligiendo el mejor partido entre las candidatas (teniendo el mismo nombre y número de parámetros; los métodos pueden ser también polimórficos, pero la mayoría de tipos específicos de sus argumentos distinguidos, determinados en tiempo de ejecución, permite la selección del método exacto para ser invocado.

2.3.Notaciones funcionales y de punto

El acceso a los atributos de tipos definidos por el usuario puede hacerse mediante dos notaciones. En muchas situaciones, las aplicaciones pueden parecer más naturales cuando se usa notación de punto.

WHERE emp.salary > 10000

Mientras en otras situaciones, una notación funcional puede ser más natural.

WHERE salary(emp) > 10000

SQL:1999 soporta las dos notaciones; de hecho, están definidas para ser variaciones sintácticas de la misma cosa.

Los métodos son un poco menos flexibles que las funciones en este caso: Solamente se puede usar notación de punto para las llamadas a métodos.

emp.salary

Un método diferente, digamos give_raise, podria combinar las dos notaciones:

Emp.give_raise(amount)

2.4.Objetos

Hemos evitado el uso de la palabra objeto hasta aquí en nuestra descripción de tipos estructurados. Esto es porque, a pesar de ciertas características como tipo de jerarquías, encapsulamiento, y un largo etc, instancias de tipos estructurados de SQL:1999 son simplemente valores, como instancias de construcción de tipos del lenguaje. Seguramente, un valor employee es mucho más complejo (tanto en apariencia como en su comportamiento) que una instancia de INTEGER.

En orden a ir ganando terreno a las características que permiten a SQL proveerse de objetos, habrá algún sentido de identificar lo que puede hacer referencia a una variedad de situaciones. En SQL:1999, esta capacidad es facilitada para permitir que los diseñadores de bases de datos puedan especificar que ciertas tablas son definidas para ser "typed tables".... esto es, sus definiciones de columna son derivadas de los atributos de un tipo estructurado.

CREATE TABLE empls OF employee

Tales tablas tienen una columna para cada atributo del tipo estructurado subyacente. ¡Las funciones, métodos y procedimientos definidos para operar en instancias del tipo, ahora operan en columnas de la tabla!. A cada columna se le dá un único identificador que se comporta exactamente como un OID (identificador de objeto), único en el espacio (esto es, dentro de la base de datos) y tiempo (la vida de la base de datos).

SQL:1999 está provisto de un tipo especial, llamado tipo REF, cuyos valores son estos identificadores únicos. Un tipo dado de REF está siempre asociado con un tipo estructurado específico. Por ejemplo, si nosotros definimos una tabla que contiene una columna llamada "manager", tales valores harán referencia a columnas en un tipo de tabla empleado, lo hace así:

Manager REF(emp_type)

En definitiva, un tipo REF es en esencia un puntero que señala a un tipo de objeto definido por el usuario.

SELECT emp.manager last_name

La notación del indicador ( ) es aplicada a un valor de algún tipo REF.