4 SQL – Un lenguaje de bases de datos relacionales

4.1 Introducción

El SQL (Structured Query Language) es un lenguaje estándar que se utiliza para definir y usar bases de datos relacionales. Originalmente se desarrolló en IBM's San Jose Research Laboratory a finales de los 70 y desde entonces ha sido adoptado y adaptado por muchos sistemas administradores de bases de datos relacionales. Ha sido aprobado como el lenguaje relacional oficial por el American National Standards Institute (ANSI) y la International Organization for Standardization (ISO).

El estándar más reciente es del año 2008 (17 de julio de 2008) y se conoce como SQL:2008. Sólo tiene algunas diferencias, no significativas en principio, con el estándar anterior SQL:1999 (también conocido como SQL3), en el cual se definieron por primera vez los conceptos del modelo objeto-relacional. La mayoría de los DBMS relacionales actuales principales (Oracle, DB2, SQL Server, Sybase, Informix) brindan las funcionalidades indicadas en este estándar; algunos, como Access, sólo proporcionan las establecidas en el estándar previo SQL-92 (del año 1992) que contempla sólo cuestiones del modelo relacional (sin objetos). En general, todos los DBMS brindan un conjunto central de facilidades definidas en el estándar, llamado SQL núcleo, agregando, o no, más facilidades según el manejador.

Este lenguaje se utiliza no sólo para hacer consultas a una base de datos, esto es para recuperar datos; sino también para definir la base de datos y los objetos que la componen, agregar, eliminar y/o modificar datos, así como para realizar otras funciones relacionadas con la manipulación de una base de datos (por ejemplo, cambiar su estructura).

El lenguaje brinda todos los elementos requeridos tanto para la programación como para la administración de las bases de datos, a fin de que cualquier usuario trabaje con una base de datos, sea éste programador de aplicaciones, administrador de las bases de datos, gerente o usuario final. Entre estos elementos se encuentran: los que definen o modifican la estructura de la base, los que manipulan la información (por ejemplo, hacer consultas), los que controlan las restricciones de integridad, los que manejan las transacciones, los que crean los grupos de usuarios, los que controlan la seguridad de la base, y muchos otros que permiten realizar casi cualquier actividad con una base de datos relacional.

El SQL no permite elaborar formas de edición de datos, reportes, menús de opciones, etc.; ya que éstas son funciones que corresponden a un lenguaje de 4a. generación. Entonces, el desarrollo normal de una aplicación completa que incluya la utilización de un DBMS de esta naturaleza implicaría, primeramente, el desarrollo de la base de datos usando este software y

luego, la elaboración de las interfaces de entrada/salida (por ejemplo, formas y reportes) con un lenguaje de 4a. generación que se pueda acoplar al DBMS (por ejemplo, Visual Basic, Java o Access). Naturalmente, este tipo de aplicaciones está relacionado con problemas que requieren de manejar volúmenes muy grandes de información; para cantidades pequeñas, basta con DBMS's para micros o estaciones de trabajo.

En estas páginas se describirán las características principales del SQL estándar.

4.2 Expresiones condicionales

Una expresión condicional es una expresión lógica que SQL evalúa a falso, verdadero o desconocido. Estas expresiones se utilizan en ciertas frases de SQL para expresar condiciones que se deben cumplir y que normalmente se relacionan con la manipulación de tuplas en relaciones. En SQL, estas expresiones también se conocen con el nombre de **predicados**. Las expresiones que se pueden expresar son las siguientes.

Expresión de comparación

Sirve para comparar dos expresiones utilizando un operador de comparación:

expresión operador_comparación expresión

donde *expresión* puede ser: constante, variable, nombre de columna, función, subconsulta, o cualquier combinación de ellos conectados por operadores aritméticos; y *operador_comparación* es uno de los operadores:

=, <>, <, <=, >, >=, !=, !>, !<

Expresión con *between*

Da un valor verdadero si *expresión1* es mayor o igual que *expresión2* y menor o igual que *expresión3*:

expresión1 [not] between *expresión2* and *expresión3*

Expresión con *in*

Da un valor verdadero si *expresión1* es igual a alguna de las expresiones que están en la lista especificada después de **in** (incluyendo subconsultas):

expresión1 [not] in {*expresión2* | ({*expresión3* | *subconsulta1*}
[,{*expresión4* | *subconsulta2*}]...)}

Expresión con *like*

Da un valor verdadero si el *patrón* especificado es encontrado en una subcadena de la primera expresión. En el *patrón* se pueden usar los caracteres especiales:

% para indicar cualquier cadena de cero o más caracteres
_ para indicar cualquier carácter

expresión [not] like *patrón*

Expresión con *null*

Da un valor verdadero si *expresión* produce un valor nulo:

expresión is [not] null

Expresión lógica

Sirve para conectar lógicamente a varias expresiones condicionales. Los operadores lógicos son los habituales: **and**, **or** y **not**. Su precedencia de evaluación, cuando aparecen en la misma expresión, es: **not**, **and** y **or**. Se pueden usar paréntesis para dar otro orden de evaluación:

[not] [(] *expre_cond* [)] [{ and | or } [not] [(] *expre_cond* [)]] ...

Expresión con *exists*

Da un valor verdadero si la tabla de resultados de una subconsulta no está vacía (esto es, contiene por lo menos una tupla):

[not] exists (*subconsulta*)

Expresión de cuantificación (operadores: *all*, *any*)

El operador **all** da un valor verdadero si *expresión operador_comparación* es cierto para todas las tuplas de la tabla de resultados de la subconsulta o si la tabla está vacía. El predicado **any** da un valor verdadero si *expresión operador_comparación* es cierto para al menos una tupla de la tabla de resultados de la subconsulta:

expresión operador_comparación { all | any } (*subconsulta*)

4.3 Definición del esquema lógico de una base de datos

4.3.1 Instrucción **create table**

Esta instrucción es la que se utiliza para definir las tablas del esquema lógico de una base de datos relacional. Cada tabla definida con esta instrucción corresponde a un esquema relacional de la base de datos; por lo tanto, deben definirse tantas tablas como esquemas relacionales se hayan obtenido previamente para la misma. Su sintaxis básica es la siguiente:

create table *nom_tabla*
(*columna_1* {*tipo_de_datos* | *dominio*} [**default** {*literal* | **null**}]
[**primary key** , **not null** , **unique** ,
references *nom_tabla_n* ,
check (*expre_cond*)]
[, *columna_2* .... , ...])

donde: *tipo_de_datos* (en SQL Server):
char(n), **smallint, int,**
real, money, datetime
literal (en SQL Server):
numéricas: 123, 34.9, 6.023E23, -0.25
cadenas: 'ejemplo'
fecha hora: '29-08-2017 10:20'

Existen variantes de algunos componentes de esta instrucción:

- Para una clave primaria de varios atributos: primary key (col1, col2, …)
- Para una clave externa de varios atributos: foreign key (col1, col2, …) references tabla
- Para un conjunto de atributos con valores únicos por tupla: unique (col1, col2, …)

4.3.2 Definición de la base de datos del mini-sistema escolar

```
--      ESQUEMA RELACIONAL DE LA BASE DE DATOS:
--              Prof(IdProf, NomProf, Categoría)
--              Alum(CU, NomAl, Carr, Prom)
--              Mater(ClaveM, NomMat, Creds)
--              Grupo(ClaveG, Salón, IdProf(FK), ClaveM(FK))
--              Inscrito(CU(FK), ClaveG(FK))
--              Historial(Folio, Calif, Fecha, CU(FK), ClaveM(FK))
--
--      DEFINICIÓN DE TABLAS
create table Prof
    (IdProf         smallint        primary key,
    NomProf         char(30),
    Categoría       char(2)         check (Categoría in ('tc','mt','tp')))
create table Alum
    (CU             smallint        primary key,
    NomAl           char(30),
    Carr            char(3)         check (Carr in ('com','ind','mat','mec','neg','tel')),
    Prom            real            check (Prom is null
                                    or Prom between 6 and 10))
create table Mater
    (ClaveM         smallint        primary key,
    NomMat          char(30),
    Creds           smallint        check (Creds between 2 and 12))
create table Grupo
    (ClaveG         char(6)         primary key,
    Salón           char(8),
    IdProf          smallint        references Prof,
    ClaveM          smallint        references Mater not null)
create table Inscrito
    (CU             smallint        references Alum,
    ClaveG          char(6)         references Grupo,
                                    primary key (CU,ClaveG))
create table Historial
    (Folio          int             primary key,
    Calif           smallint        check (Calif between 5 and 10),
    Fecha           datetime        check (Fecha > '1990-01-01'),
    CU              smallint        references Alum not null,
    ClaveM          smallint        references Mater not null)
```

4.3.3 Creación genérica de tablas

En esta parte se muestra la creación genérica de tablas de los diversos conceptos de entidad-vínculo y modelo relacional, tomando como base los diagramas y esquemas relacionales de la sección 3.4 del capítulo 3 de estas notas. En los campos de las tablas se usan diversos tipos de datos simplemente para mostrar las alternativas que brinda SQL para aquellos.

1. Vínculo 1-1

Alternativa 1	**Alternativa 2**
create table A (A1 int primary key, A2 real, -- Vínculo 1-1: B1 int references B unique); -- Nota: primero se debe crear B y -- luego A. create table B (B1 int primary key, B2 real);	create table A (A1 int primary key, A2 real); create table B (B1 int primary key, B2 real, -- Vínculo 1-1: A1 int references A unique);

2. Vínculo 1-N

create table A (A1 int primary key, A2 date);	create table B (B1 int primary key, B2 money, -- Vínculo 1-N: A1 int references A);

3. Vínculo M-N

create table A (A1 int primary key, A2 char(30)); create table B (B1 int primary key, B2 varchar(30));	-- Vínculo M-N: create table R (A1 int references A, B1 int references B, primary key (A1, B1), -- Puede haber atributos adicionales: R1 smallint);

4. Vínculo ISA

-- Tipo base: create table A (A1 int primary key, A2 real, -- Atributos comunes. Tipo char(3)); -- Tipo derivado: create table B (A1 int primary key references A, B2 real, -- Atributos diferentes. B3 date);	-- Tipo derivado: create table Y (A1 int primary key references A, Y2 date, -- Atributos diferentes. Y3 varchar(30));

5. Entidad débil

```
-- Tipo dueño:
create table A
(A1      int        primary key,
 A2      real);
```

```
-- Tipo dependiente (débil):
create table B
(A1      int        references A,
 B1      int,
                    primary key (A1, B1),
 B2      money);
```

6. Vínculos recursivos

```
-- Vínculo 1-1:
create table A
(A1      int        primary key,
 A2      char(20),
 A1Bis   int        references A unique);
```

```
-- Vínculo 1-N:
create table A
(A1      int        primary key,
 A2      varchar(20),
 A1Bis   int        references A);
```

```
create table A
(A1      int        primary key,
 A2      real);
```

```
-- Vínculo M-N:
create table R
(A1      int        references A,
 A1Bis   int        references A,
                    primary key (A1, A1Bis),
-- Puede haber atributos adicionales:
 R1      money);
```

7. Vínculo ternario (sólo dos casos)

```
create table A
(A1      int        primary key,
 A2      char(30));

create table B
(B1      int        primary key,
 B2      varchar(30));

create table C
(C1      int        primary key,
 C2      date);
```

```
-- Vínculo 1-N-1 (un caso):
create table R
(A1      int        references A,
 B1      int        references B,
 C1      int        references C,
                    primary key (A1, B1),
 -- Puede haber atributos adicionales:
 R1      smallint);


-- Vínculo L-M-N:
create table R
(A1      int        references A,
 B1      int        references B,
 C1      int        references C,
                    primary key (A1, B1, C1),
 -- Puede haber atributos adicionales:
 R1      int);
```

4.4 Instrucciones para actualizar la base de datos

4.4.1 Instrucción **insert**

Esta instrucción se utiliza para agregar tuplas a una tabla. Su sintaxis (básica) es:

insert into *nom_tabla* [(*lista_de_columnas*)]
values (*lista_de_valores*)

donde:
lista_de_columnas: representa a las columnas de la tabla (separadas por coma).
lista_de_valores: representa los valores de la tupla a insertar.

Ejemplos:

```
insert into Grupo (ClaveG, Salón,ClaveM)
   values ('300-1','CC102',300)

insert into Prof
   values (5, 'Raúl', 'mt')
```

4.4.2 Instrucción **update**

Sirve para modificar los valores de las columnas de una tabla. Su sintaxis (básica) es:

update *nom_tabla*
set *columna_1 = valor_1* [, *columna_2 = valor_2*, ...]
[where *expre_cond*]

donde:
columna_1, *columna_2*, ...: son las columnas a modificar.

Ejemplos:

```
update Historial set Calif=8
   where CU=20 and ClaveM=620
```

4.4.3 Instrucción **delete**

Se emplea para eliminar tuplas de una tabla. Su sintaxis (básica) es:

delete from *nom_tabla*
[**where** *expre_cond*]

Ejemplos:

```
delete from Inscrito where CU = 60

delete from Inscrito
```

4.5 Instrucción select

SQL tiene una instrucción básica para recuperar información de una base de datos: la instrucción **select**. Esta instrucción genera una tabla de resultados que contiene las tuplas que produce la consulta. La instrucción presenta diversas opciones las cuales se discutirán gradualmente en las siguientes páginas.

La sintaxis general de la instrucción select es:

select [**all** | **distinct**] *lista_de_columnas_s*
from *lista_de_tablas*
[**where** *expre_cond*]
[**group by** *lista_de_columnas_g*]
[**having** *expre_cond*]
[**order by** {*columna* | *posición_de_columna*}
[**asc** | **desc**]...]

donde:

lista_de_columnas_s: es una lista de columnas (separadas con coma) cuyos valores van a ser recuperados por la consulta. Una columna puede ser: [{*nombre_de_tabla* | *alias*}.]*nombre_de_columna*,
[{*nombre_de_tabla* | *alias*}.]* o *expresión*

lista_de_tablas: es una lista de tablas (separadas con coma) requeridas para procesar la consulta

expre_cond: es uno de los predicados que pueden usarse en SQL

lista_de_columnas_g: es una lista de columnas (separadas con coma); para cada columna sólo se puede usar la sintaxis:
[{*nombre_de_tabla* | *alias*}.]*nombre_de_columna*,

columna: es de la forma: [{*nombre_de_tabla* | *alias*}.]*nombre_de_columna*

posición_de_columna: es un número que indica la posición de una columna en la tabla de resultados.

Cada una de las distintas cláusulas de la instrucción select genera una tabla intermedia que es usada para evaluar la cláusula siguiente.

La cláusula **select** especifica las columnas cuyos valores se van a tomar para generar la tabla de resultados.

La cláusula **from** especifica las tablas (relaciones) de donde se van a tomar las columnas indicadas en la cláusula select.

La cláusula **where** sirve para especificar una condición que deben cumplir las tuplas que se seleccionen de las tablas indicadas en from.

La cláusula **group by** sirve para formar grupos con las tuplas que aparecen en la tabla intermedia producida por where o por from.

La cláusula **having** se emplea para aplicar algún predicado a los grupos formados con group by.

Finalmente, la cláusula **order by** sirve para ordenar las tuplas de la tabla de resultados.

El DBMS evalúa las cláusulas en la instrucción select en el siguiente orden: *from*, *where*, *group by*, *having*, *lista_de_columnas_s*, *order by*. Como se mencionó anteriormente, después de hacer una evaluación dada, conceptualmente se puede considerar que el DBMS produce una tabla intermedia la cual es usada para realizar la siguiente evaluación, y así hasta finalizar la ejecución de la instrucción.

Opcionalmente se puede usar la cláusula **union** para producir una sola tabla uniendo el resultado de varias instrucciones select. La sintaxis es la siguiente:

select_1
union [**all**]
select_2
[**union** [**all**]...]
[**order by** *resto_del_order_by*]

Por default, el resultado de union elimina tuplas duplicadas. Para conservar todas las tuplas resultantes de la unión se debe usar all. Cuando se usa union las instrucciones select_1, select_2, ... no deben contener order by.

Observaciones sobre las cláusulas que componen a *select*

- Cláusula *where*

El DBMS evalúa la expresión condicional dada en *where* para cada tupla de la tabla intermedia creada por *from.*
Las columnas especificadas en la expresión condicional de *where*:

- deben ser columnas de la tabla intermedia creada por *from*, o
- pueden ser una referencia externa (una *referencia externa* es una referencia dentro de una subconsulta a una tabla declarada en una (sub)consulta externa que englobe a dicha subconsulta). Por ejemplo, en la siguiente consulta

```
select  NomAl
from  Alum
where not  exists
    (select * from Inscrito where Alum.CU = Inscrito.CU)
```

Alum, dentro de la subconsulta, es una referencia externa con respecto a dicha subconsulta, ya que la tabla está declarada en la consulta externa.

- Cláusula *group by*

Especifica columnas que el DBMS usa para formar grupos con la tabla intermedia creada por *where*, si existe, o por *from.* Se pueden especificar varias columnas consecutivas, con lo cual se van formando subgrupos dentro de los grupos. Ejemplo:

```
select NomProf, g.ClaveG, count(*) Cant_Alumnos
from Prof p, Grupo g, Inscrito i
where p.IdProf=g.IdProf and g.ClaveG = i.ClaveG
group by NomProf, g.ClaveG
```

En este ejemplo se forman grupos por cada *profesor* que imparte cursos y, para cada uno, se forman subgrupos con las *claves de los grupos* en los cuales están inscritos los alumnos. Todos los valores nulos para una columna dada son agrupados juntos.

Cuando se usa esta cláusula, columnas que aparecen en *lista_de_columnas_s*, no especificadas en *group by*, deben colocarse siempre dentro de una función. En caso contrario, no pueden aparecer en dicha lista. Por ejemplo, la siguiente consulta es incorrecta

```
select  Carr, NomAl, count(*)
from Alum
group by Carr
```

debido a que *NomAl* es una columna que no aparece dentro de *group by*, ni dentro de una función.

- Cláusula *having*

El DBMS aplica la expresión condicional de *having* a cada grupo de la tabla intermedia creada por la cláusula precedente (que en general es *group by*).

La cláusula *having* afecta grupos de la misma forma que *where* afecta tuplas individuales.
Si *having* no es precedida por *group by*, el DBMS aplica la expresión condicional a todas las tuplas de la tabla intermedia creada por *where* o por *from*, como si se tratará de un solo grupo.

Las columnas especificadas en *having* deben:
- estar también en *group by*,
- ser especificadas dentro de una función, o

- Subconsultas

Cualquier consulta anidada sólo puede generar como resultado una tabla con **una sola** columna (la cual, por supuesto, puede contener varios valores). Cuando se tienen varias consultas anidadas, el orden de evaluación es de la más interna hacia la más externa.

Restricciones de uso

- En general, el predicado en *where* no puede contener una función de totales. La única excepción a esta restricción es cuando la función en *where* tiene como argumento una referencia externa.
- Las subconsultas sólo pueden entregar una tabla de una sola columna, aunque puede haber muchos valores en ésta.
- Si **no** se usa *group by*, la *lista_de_columnas_s* de la cláusula *select* debe:
  - ser únicamente una lista de funciones de totales, o
  - ser únicamente una lista de atributos de tablas.

  Por lo anterior, el siguiente ejemplo es incorrecto:

  ```
  select Carr, avg(Prom)
  from Alum
  ```

  dado que se está combinando una columna, *Carr*, con una función de totales, *avg(Prom)*, sin utilizar la cláusula *group by*.

- Si se usa *group by*, las columnas en la *lista_de_columnas_s* deben:
  - estar también especificadas en *group by*, o
  - ser especificadas dentro de una función de totales.
- Las funciones de totales, en general, no pueden estar anidadas. Es decir, normalmente no se debería escribir: count(sum(Creds)), aunque hay DBMS's que sí lo aceptan.

4.6 Eliminación de objetos de una base de datos

Para eliminar algún objeto de una base de datos se utiliza la instrucción **drop**. En particular, cuando se elimina una tabla desaparecen su estructura, los datos que almacena, así como los índices y permisos que tiene asociados. La sintaxis de la instrucción es:

Para tablas:

drop table *nombre_tabla* [, *nombre_tabla*] ...