Unidad 3: BBDD relacionales

BBDD01, Sesión 7: SQL

INDICE

- Introducción
- Estructura básica
- Operaciones sobre conjuntos
- Funciones de agregación
- Valores nulos
- Subconsultas anidadas
- Vistas
- Consultas complejas
- Modificación de la base de datos
- Lenguaje de definición de datos
- Otras características SQL

Referencias: Silberschatz 4ª Ed. pp 87-118

Introducción

- Lenguaje de consulta más cómodo al usuario (no navegable)
- Basado en algebra relacional + cálculo relacional
- Versión original ⇒ IBM (Sequel dentro de System R)
- Evolucionó ⇒ Structured Query Language
- Estandarización ⇒ ANSI e ISO
  - Normas 1986: SQL-86
  - Normas: SQL 89, SQL 92 y SQL 99
- Componentes:
  - LDD
  - LMD
  - Vistas
  - Transacciones
  - SQL incorporado y dinámico
  - Integridad
  - Autorización.

Introducción

- Ejemplos basados en Empresa bancaria:

Esquema-sucursal = (*nombre-sucursal*, *ciudad-sucursal*, *activo*)
Esquema-cliente = (*nombre-cliente*, *calle-cliente*, *ciudad-cliente*)
Esquema-préstamo = (*número-préstamo*, *nombre-sucursal*, *importe*)
Esquema-prestatario = (*nombre-cliente*, *número-préstamo*)
Esquema-cuenta = (*número-cuenta*, *nombre-sucursal*, *saldo*)
Esquema-impositor = (*nombre-cliente*, *número-cuenta*)

Estructura básica

- Operación estrella:

Select … from … Where …

- Select ⇒ hacer una consulta
- From ⇒ tablas en la que se solicita información
- Where ⇒ selección sobre lo descrito en *from*

$$\textbf{select } A_1, A_2, \ldots, A_n$$
$$\textbf{from } r_1, r_2, \ldots, r_m$$
$$\textbf{where } P$$

- Donde A_i son los atributos, r_i las tablas (relaciones) y P condición
- Si se omite where devuelve toda la información.

Estructura básica

- Siempre el resultado es una nueva tabla
- Cláusula Select

```
select nombre-sucursal
from préstamo
```

- Trabaja con duplicados por defecto ⇒ eliminarlos es costoso
- Se puede con **distinct**:

```
select distinct nombre-sucursal
from préstamo
```

- Permite usar **all** ⇒ indicar que no se eliminan duplicados

```
select all nombre-sucursal
from préstamo
```

Estructura básica

- * $\Rightarrow$ indica todos los atributos
  - Select * , indica todos los atributos de todas las relaciones
  - Select prestamo.* , indica todos los atributos de la relación prestamo
- Puede haber expresiones aritméticas +,-,*,/

```
select nombre-sucursal, número-préstamo,
       importe * 100
from préstamo
```

- Cláusula where

```
select número-préstamo
from préstamo
where nombre-sucursal = 'Navacerrada' and
      importe > 1200
```

- Operadores lógicos AND,OR, NOT
- Operadores de comparación: >,<,=,>=,<=, <>

Estructura básica

- Operador comparación **between** y **not between**

```
select número-préstamo
from préstamo
where importe between 90000 and 100000
```

- Cláusula from

- Realiza el producto cartesiano de las relaciones

```
select nombre-cliente, prestatario.número-préstamo,
       importe
from prestatario, préstamo
where prestatario.número-préstamo
      = préstamo.número-préstamo
```

Estructura básica

- Operación renombramiento
- Cláusula **as** tanto en Select como en from

nombre-antiguo **as** *nombre-nuevo*

```
select nombre-cliente, prestatario.número-préstamo
       as id-préstamo, importe
from prestatario, préstamo
where prestatario.número-préstamo =
      préstamo.número-préstamo
```

- Definición de variables tupla (opcional el as):

```
select nombre-cliente, T.número-préstamo, S.importe
from prestatario as T, préstamo as S
where T.número-préstamo = S.número-préstamo
```

```
select distinct T.nombre-sucursal
from sucursal as T, sucursal as S
where T.activo > S.activo and S.ciudad-sucursal
      = 'Barcelona'
```

Estructura básica

- Operaciones sobre cadenas
- Cadenas entre comillas simples: ‘Navacerrada’
- Comparación con patrones: **like**
  - %, encaja con cualquier cadena
  - _ , encaja con cualquier carácter
  - Ejemplo: ‘Nava%’, ‘%cer%’, ‘_ _ _’, ‘_ _ _ %’

```
select nombre-cliente
from cliente
where calle-cliente like ‘%Mayor%’
```

- Carácter de escape: palabra escape

```
like ‘ab\%cd%’ escape ‘\’        like ‘ab\\cd%’ escape ‘\’
```

- Not like, similar to, concatenación de caracteres (||)

Estructura básica

- Ordenación de tuplas
- Ordenación tuplas resultantes: order by
  - Asc, indica ascendente (defecto)
  - Desc, indica descendente

```
select distinct nombre_cliente
from prestatario, préstamo
where prestatario.número-préstamo
        = préstamo.número-préstamo and
        nombre-sucursal = 'Navacerrada'
order by nombre-cliente
```

- Ordenar por varios atributos

```
select *
from préstamo
order by importe desc, número-préstamo asc
```

Operaciones sobre conjuntos

- SQL-92:
  - Intersección ⇒ intersects
  - Unión ⇒ union
  - Diferencia ⇒ except
- Las relaciones deben de ser compatibles
- Operación unión

```
(select nombre-cliente
 from impositor)
 union
 (select nombre-cliente
  from prestatario)
```

- Elimina duplicados por defecto. **Union all** los conserva

```
(select nombre-cliente
 from impositor)
union all
(select nombre-cliente
 from prestatario)
```

Operaciones sobre conjuntos

- Operación intersección

```
(select distinct nombre-cliente
 from impositor)
intersect
(select distinct nombre-cliente
 from prestatario)
```

- Elimina duplicados por defecto. Con **all** se mantienen
- Operación diferencia

```
(select distinct nombre-cliente
 from impositor)
except
(select distinct nombre-cliente
 from prestatario)
```

- Elimina duplicados automáticamente. Con **all** se mantienen

Funciones agregadas

- Toman una colección de valores de entrada y devuelven uno de salida:
  - Media: avg
  - Mínimo: min
  - Máximo: max
  - Suma: sum
  - Cuenta: count
- Sum y avg tienen que ser sobre datos numéricos

```
select avg (saldo)
from cuenta
where nombre-sucursal = 'Navacerrada'
```

- Se pueden aplicar a un grupo de conjuntos de tuplas: group by

```
select nombre-sucursal, avg (saldo)
from cuenta
group by nombre-sucursal
```

Funciones agregadas

- Si se desea eliminar los duplicados antes de efectuar la agregación ⇒ distinct

```
select nombre-sucursal, count (distinct nombre-
       cliente)
from impositor, cuenta
where impositor.número-cuenta = cuenta.número-
      cuenta
group by nombre-sucursal
```

- Condiciones a cada uno de los grupos ⇒ having

```
select nombre-sucursal, avg (saldo)
from cuenta
group by nombre-sucursal
having avg (saldo) > 1200
```

- Contar tuplas de una relación ⇒ count(*). No permite distinct

```
select count (*)
from cliente
```

Funciones agregadas

- Ejemplo:

```
select impositor.nombre-cliente, avg (saldo)
from impositor, cuenta, cliente
where impositor.número-cuenta
        = cuenta.número-cuenta and
        impositor.nombre-cliente
        = cliente.nombre-cliente and
        ciudad-cliente = 'Madrid'
group by impositor.nombre-cliente
having count (distinct impositor.número-cuenta) >= 3
```

- Las funciones agregadas no se pueden componer: max(avg (…)) no está permitido

Valores nulos

- Modelo relacional permite valores nulos ⇒ no hay información
- Palabra NULL para buscar esa información (is null/is not null)

select *número-préstamo*
from *préstamo*
where *importe* **is null**

- Expresión aritmética que contenga null ⇒ null
- Comparaciones que contengan null ⇒ null (desconocido)
  - Null no es igual a null, ni distinto de null.
- Con operadores lógicos:

- **and**: el resultado de *cierto* **and** *desconocido* es *desconocido*, *falso* **and** *desconocido* es *falso*, mientras que *desconocido* **and** *desconocido* es *desconocido*.
- **or**: el resultado de *cierto* **or** *desconocido* es *cierto*, *falso* **or** *desconocido* es *desconocido*, mientras que *desconocido* **or** *desconocido* es *desconocido*.

Valores nulos

- Si el predicado de where es desconocido ⇒ no añade tupla
- Is unknown ó is not unknown ⇒ comprobar si el resultado de una comparación es desconocido ó no
- Operaciones de agregación ⇒ ignora los valores nulos salvo la función count(*)
  - Si todos son nulos, devuelve 0
  - Las demás funciones devuelven vacio
- En SQL-99
  - Tipo de dato boolean. Cierto, falso y desconocido
  - Funciones de agregación: some, every

Subconsultas anidadas

- Es una expresión select-from-where que se anida dentro de otra consulta (detrás del from, en vez de una tabla va otro select)
- Uso:
  - Comprobación de pertenencia a conjuntos
  - Comparación de conjuntos
  - Cardinalidad de conjuntos
- Pertenencia a conjuntos
- Utiliza cálculo relacional para la pertenencia a conjuntos
- **(not) in**

```
select distinct nombre-cliente
from prestatario
where nombre-cliente in (select nombre-cliente
from impositor)
```

Subconsultas anidadas

- Se puede usar con más de un atributo

```
select distinct nombre-cliente
from prestatario, préstamo
where prestatario.número-préstamo =
        préstamo.número-préstamo and
        nombre-sucursal = 'Navacerrada' and
        (nombre-sucursal, nombre-cliente) in
           (select nombre-sucursal, nombre-cliente
           from impositor, cuenta
           where impositor.número-cuenta
                   = cuenta. número-cuenta)
```

```
select distinct nombre-cliente
from prestatario
where nombre-cliente not in (select nombre-cliente
                             from impositor)
```

- Sobre conjuntos enumerados

```
select distinct nombre-cliente
from prestatario
where nombre-cliente not in ('Santos', 'Gómez')
```

Subconsultas anidadas

- Operaciones de comparación

```
select distinct T.nombre-sucursal
from sucursal as T, sucursal as S
where T.activo > S.activo and
      S.ciudad-sucursal = 'Barcelona'
```

- Utilizar some (al menos). Antiguamente ANY

```
select nombre-sucursal
from sucursal
where activo > some (select activo
                     from sucursal
                     where ciudad-sucursal
                           = 'Barcelona')
```

- Se puede usar con: <, <=,>,>=,<>

Subconsultas anidadas

- Para comparar con todas las tuplas: all

```
select nombre-sucursal
from sucursal
where activo > all (select activo
                    from sucursal
                    where ciudad-sucursal
                          = 'Barcelona')
```

- Se puede usar con: <, <=,>,>=,<>

```
select nombre-sucursal
from cuenta
group by nombre-sucursal
having avg (saldo) >= all (select avg (saldo)
                           from cuenta
                           group by nombre-sucursal)
```

Subconsultas anidadas

- Comprobación de relaciones vacías
- Si se devuelven tuplas o no ⇒ exists (true si no es vacia)

```
select nombre-cliente
where exists (select *
              from impositor
              where impositor.nombre-cliente =
                    prestatario.nombre-cliente)
```

- Not exists ⇒ inexistencia de tuplas

```
select distinct S.nombre-cliente
from impositor as S
where not exists ((select nombre-sucursal
                   from sucursal
                   where ciudad-sucursal
                         = 'Barcelona')
                  except
                   (select R.nombre-sucursal
                    from impositor as T, cuenta as R
                    where T.número-cuenta
                          = R.número-cuenta and
                          S.nombre-cliente
                          = T.nombre-cliente ))
```

Subconsultas anidadas

- Comprobación de tuplas duplicadas
- Unique ⇒ cierto si no devuelve tuplas duplicadas

```
select T.nombre-cliente
from impositor as T
where unique (select R.nombre-cliente
              from cuenta, impositor as R
              where T.nombre-cliente
                    = R.nombre-cliente and
                    R.número-cuenta
                    = cuenta.número-cuenta and
                    cuenta.nombre-sucursal
                    = 'Navacerrada')
```

- Not unique ⇒ si hay tuplas duplicadas

Vistas

- Se define con **create view**

create view *v* **as** <expresión de consulta>

- Ejemplo:

```
create view todos-los-clientes as
    (select nombre-sucursal, nombre-cliente
     from impositor, cuenta
     where impositor.número-cuenta
           = cuenta.número-cuenta)
  union
    (select nombre-sucursal, nombre-cliente
     from prestatario, préstamo
     where prestatario.número-préstamo
           = préstamo.número-préstamo)
```

Vistas

- Se pueden redefinir los nombres de los atributos

```
create view total-préstamos-sucursal
        (nombre-sucursal, total-préstamos) as
select nombre-sucursal, sum (importe)
from préstamo
group by nombre-sucursal
```

- Después se utiliza como cualquier otra relación:

```
select nombre-cliente
from todos-los-clientes
where nombre-sucursal = 'Navacerrada'
```

Consultas complejas

- Varios bloques SQL dentro de una consulta:
- Relaciones derivadas
- Subconsulta dentro de la cláusula from

```
select nombre-sucursal, saldo-medio
from (select nombre-sucursal, avg (saldo)
        from cuenta
        group by nombre-sucursal)
      as resultado (nombre-sucursal, saldo-medio)
where saldo-medio > 1200
```

```
select max(saldo-total)
from (select nombre-sucursal, sum(saldo)
        from cuenta
        group by nombre-sucursal) as
              total-sucursal(nombre-sucursal,
              saldo-total)
```

Consultas complejas

- Cláusula with
- Introducida en SQL-99. No lo incorporan todos los SGBD
- Define una vista temporal que existe mientras exista la consulta

```
with saldo-máximo(valor) as
        select max (saldo)
        from cuenta
select número-cuenta
from cuenta, saldo-máximo
where cuenta.saldo = saldo-máximo.valor
```

Modificación de la base de datos

- Borrado
- Se expresa como una consulta
- Donde r es una tabla y P el predicado
- Si se omite el predicado ⇒ se borran todas las tuplas

```
delete from r
where P
```

```
delete from préstamo
```

```
delete from cuenta
where nombre-sucursal = 'Navacerrada'
```

```
delete from cuenta
where nombre-sucursal in (select nombre-sucursal
                          from sucursal
                          where ciudad-sucursal
                                = 'Navacerrada')
```

Modificación de la base de datos

- Inserción
- Se inserta o bien la tupla deseada o el resultado de una consulta:
  - Debe de respetar el dominio de los atributos
  - Y el número de atributos
- Se usa la clausula insert

insert into *cuenta*
values ('C-9732', 'Navacerrada', 1200)

- Se puede especificar el orden de los atributos

insert into *cuenta* (*nombre-sucursal*, *número-cuenta*, *saldo*)
values ('Navacerrada', 'C-9732', 1200)

Modificación de la base de datos

- Se pueden insertar tuplas provenientes de una consulta

```
insert into cuenta
        select nombre-sucursal, número-préstamo, 200
        from préstamo
        where nombre-sucursal = 'Navacerrada'
```

- Importante ⇒ finalice la sentencia select antes de insertar

```
insert into cuenta
        select *
        from cuenta
```

- Se pueden insertar valores nulos: null

```
insert into cuenta
        values ('C-401', null, 1200)
```

Modificación de la base de datos

- Actualización
- Actualizar algunos campos de las tuplas ⇒ UPDATE

```
update cuenta
set saldo = saldo * 1.05
```

```
update cuenta
set saldo = saldo * 1.05
where saldo >= 1000
```

- Constructor CASE

```
case
    when pred1 then result1
    when pred2 then result2
    ...
    when predn then resultn
    else result0
end
```

```
update cuenta
set saldo = case
                when saldo <= 10000 then saldo * 1.05
                else saldo * 1.06
            end
```

Modificación de la base de datos

- Actualización sobre vistas

- Pueden producir problemas si la vista no contiene la kp de la tabla original
- Muchas bases de datos imponen:

> Una modificación de una vista es válida sólo si la vista en cuestión se define en términos de la base de datos relacional real, esto es, del nivel lógico de la base de datos (y sin usar agregación).

- En general insert, update y delete estan prohibidos en vistas

Modificación de la base de datos

- Transacción
- Secuencia de instrucciones SQL que se ejecuta como una operación atómica
  - Ej: trasfiere 500€ de la cuenta 300 a la 301
  - O está hecha del todo, o no hecha en absoluto
- SQL una transacción comienza implícitamente cuando se ejecuta una instrucción implícitamente:
  - COMMIT WORK, finaliza la transacción anotando los cambios en la BD
  - ROLLBACK WORK, deshace los cambios en la BD
- SQL-99.
  - Cada instrucción SQL es una transacción implícitamente.
  - Se especifica una transacción con BEGIN, END

Reunión de Relaciones

- Reunión interna

préstamo **inner join** *prestatario* **on**
préstamo.número-préstamo
= prestatario.número-préstamo

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*	*número-préstamo*
P-170	Centro	3.000	Santos	P-170
P-230	Moralzarzal	4.000	Gómez	P-230

- Reunión externa

préstamo **left outer join** *prestatario* **on**
préstamo.número-préstamo
= prestatario.número-préstamo

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*	*número-préstamo*
P-170	Centro	3.000	Santos	P-170
P-230	Moralzarzal	4.000	Gómez	P-230
P-260	Navacerrada	1.700	*null*	*null*

Reunión de Relaciones

- Reunión natural

préstamo **natural inner join** *prestatario*

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez

- Tipos y condiciones de reunión

Tipos de reunión
inner join
left outer join
right outer join
full outer join

Condiciones de reunión
natural
on <predicado>
using (A_1, A_2, …, A_n)

- Condición de reunión obligatoria en la externa
- Reunión natural: primero los atributos de reunión + atributos de la relación izquierda + atributos de relación derecha

Reunión de Relaciones

- SQL-92 otros 2 tipos de reunión:
  - Cross join ⇒ reunión cruzada (sin condición de reunión)
  - Union join ⇒ reunión de unión (reunión externa completa con condición falsa)

Lenguaje de definición de datos

- El LDD permite especificar:
  - Esquema de cada relación
  - El dominio de valores asociado a cada atributo
  - Restricciones de integridad
  - Índices asociados a cada relación
  - Información de seguridad y autorización
  - Estructura de almacenamiento físico en disco.
- Tipos de dominio base
  - Char(n), varchar(n),text
  - int, integer, smallint, float, real, double precission,numeric(p,d)
  - Date, time, timestamp, función extract(campo from d), interval
  - Valor null , pertenece a todos los dominios base
  - Se puede especificar que un dominio es not null

número-cuenta **char**(10) **not null**

Lenguaje de definición de datos

- Definición de esquemas
- Orden create table

create table r (A_1D_1, A_2D_2, … A_nD_n,
⟨restricción-integridad$_1$⟩.
…
⟨restricción-integridad$_k$⟩)

- Donde A_i es el nombre atributo y D_i el dominio
- Restricción de integridad:
  - Primary key (A_1,A_2,…,A_n), valores no nulos y únicos (Opcional)
  - Check (P) , predicado P que debe de satisfacer el atributo de la tupla

create table *cuenta*
(*número-cuenta* **char** (10),
nombre-sucursal **char** (15),
saldo **integer**,
primary key (*número-cuenta*),
check (*saldo* >= 0))

  - Unique (A_1,A_2,…,A_n) , especificar clave candidata. Permite nulos.

Lenguaje de definición de datos

- Borrar esquema de tabla ⇒ drop table

  drop table r

  - Borra las tuplas y la relación

- SQL-92 ⇒ alter table. Permite modificar un esquema de tabla
  - Añadir atributo:

    alter table r **add** A D

  - Eliminar atributo:

    alter table r **drop** A

Otras características

- SQL incorporado
- Utilización de SQL dentro de lenguaje de programación: C, java, fortran, pascal, etc. (lenguaje anfitrión)
- Las estructuras que se incluye y permiten ⇒ SQL incorporado
- Se necesita un preprocesador

```
EXEC SQL <instrucción de SQL incorporado>
  END-EXEC

# SQL { <instrucción de SQL incorporado> };
```

  - Java (SQLJ):

```
EXEC SQL
        declare c cursor for
        select nombre-cliente, ciudad-cliente
        from impositor, cliente
        where impositor.nombre-cliente
                = cliente.nombre-cliente and
                cuenta.número-cuenta
                = impositor.número-cuenta and
                impositor.saldo > :importe
END-EXEC
```

Otras características

- SQL dinámico
- Permite construir y ejecutar consultas en tiempo de ejecución

```
char * prog_sql =«update cuenta set saldo
                 = saldo * 1.05
                 where número-cuenta = ?»
EXEC SQL prepare prog_din from :prog_sql;
char cuenta[10] = «C-101»;
EXEC SQL execute prog_din using :cuenta;
```

- Necesita extensiones del lenguaje y un preprocesador
- Mejor ⇒ normas de conexión. Interfaces para programas de aplicación.
  - Norma ODBC (Open Database Conectivity) en C
  - Norma JDBC en Java

Otras características

- ODBC

```
int ODBCexample()
{
    RETCODE error;
    HENV ent; /* entorno */
    HDBC con; /* conexión a la base de datos */

    SQLAllocEnv(&ent);
    SQLAllocConnect(ent, &con);
    SQLConnect(con, «aura.bell-labs.com», SQL NTS, «avi», SQL NTS, «avipasswd», SQL NTS);
    {
        char nombresucursal[80];
        float saldo;
        int lenOut1, lenOut2;
        HSTMT stmt;

        char * consulta = «select nombre_sucursal, sum (saldo)
                           from cuenta
                           group by nombre_sucursal»;
        SQLAllocStmt(con, &stmt);
        error = SQLExecDirect(stmt, consulta, SQL NTS);
        if (error == SQL SUCCESS) {
            SQLBindCol(stmt, 1, SQL C CHAR, nombresucursal, 80, &lenOut1);
            SQLBindCol(stmt, 2, SQL C FLOAT, &saldo, 0 , &lenOut2);
            while (SQLFetch(stmt) >= SQL SUCCESS) {
                printf (« %s %g\n», nombresucursal, saldo);
            }
        }
        SQLFreeStmt(stmt, SQL DROP);
    }
    SQLDisconnect(con);
    SQLFreeConnect(con);
    SQLFreeEnv(ent);
}
```

Otras características

- JDBC

```
public static void ejemploJDBC (String idbd, String idusuario, String contraseña)
{
    try
    {
        Class.forName («oracle.jdbc.driver.OracleDriver»);
        Connection con = DriverManager.getConnection
                («jdbc:oracle:thin:@aura.bell-labs.com:2000:bdbanco»,
                idusuario, contraseña);
        Statement stmt = con.createStatement();
        try {
            stmt.executeUpdate(
                «insert into cuenta values('C-9732', 'Navacerrada', 1200)»);
        } catch (SQLException sqle)
        {
            System.out.println(«No se pudo insertar la tupla. » + sqle);
        }
        ResultSet rset = stmt.executeQuery
                («select nombre_sucursal, avg (saldo)
                from cuenta
                group by nombre_sucursal»);
        while (rset.next()) {
            System.out.println(rset.getString(«nombre_sucursal») + « » +
                    rset.getFloat(2));
        }
        stmt.close();
        con.close();
    }
    catch (SQLException sqle)
    {
        System.out.println(«SQLException : » + sqle);
    }
}
```

Otras características

- Bases de datos formados por:
  - Catálogos
  - Esquemas
  - Objetos: relaciones, vistas
- Cada usuario tiene un catálogo asignado

catálogo5.esquema-banco.cuenta

- Puede crear esquemas y borrarlos: create (drop) scheme
- Extensiones procedimentales (SQL-92)
  - Crear procedimientos (begin, end)
    - Nombre
    - Parámetros de entrada
    - Conjunto de instrucciones SQL
  - Procedimientos almacenados