BASES DE DATOS
TEMA 4. SQL. UN LENGUAJE DE CONSULTA COMERCIAL PARA BASES DE DATOS RELACIONALES

Contenidos generales

* Definición de datos en SQL
* Consulta de datos en SQL
  - Estructura básica de una sentencia SQL
  - Operaciones de conjuntos
  - Funciones de agregación
  - Vistas
* Actualización de datos en SQL

Motivación

BD deben facilitar la definición y recuperación de datos
Las BDR se encuentran bien establecidas
SQL es un lenguaje de consulta estándar para BDR
- Definición de datos
- Consulta de datos
- Actualización de datos

4.1. Introducción (1)

Algebra relacional como lenguaje de consulta formal procedimental -> Especificar el cómo
SGBDRs comerciales ofrecen una interfaz SQL
Sentencias SQL en aplicaciones (*SQL embebido*)

SQL permite definir la base de datos así como consultar y modificar sus datos

4.1. Introducción (2)

Sucursales

nombreSuc	ciudadSuc	Activo
Downtown	Brooklyn	9000000
Redwood	Palo Alto	2100000
Perrydge	Horseneck	1700000
Mianus	Horseneck	400000
Round Hill	Horseneck	8000000
Pownal	Bennington	300000
North Town	Rye	3700000
Brighton	Brooklyn	7100000

Empleados

nombreEmp	dniEmp	telefono	NombreSuc
Smith	10	101010	Downtown
Kortz	11	111111	Downtown
Hansen	12	121212	Perrydge
Dubitzky	13	131313	Perrydge
Henson	14	141414	Mianus
Kravitz	15	151515	Brighton

Cuentas

numeroCta	saldo	nombreSuc
1	10000	Downtown
2	20000	Downtown
3	30000	Perrydge
4	40000	Perrydge
5	50000	Mianus
6	60000	Brighton

Clientes

nombreCli	dniCli	Domicilio
Johnson	1	La Reina nº7
Smith	2	Fragata azul nº8
Hayes	3	Gibraltar español nº14
Turner	4	Gibraltar español nº17
Williams	5	Diamante S/N
Lindsay	6	Gato negro nº13
Green	7	Perro nº1

CtaCli

dniCli	numeroCta
1	1
1	2
2	3
3	4
4	5
5	5
6	5
7	6

Transacciones

numeroCta	numeroTrans	fecha	importe
1	1	10-10	+10000
2	1	10-10	+30000
2	2	11-10	-20000
3	1	12-10	+30000
4	1	12-10	+40000
5	1	13-10	+50000
6	1	13-10	+60000

4.2. Definición de datos en SQL (1)

Definición de datos en SQL: Creación, modificación y eliminación de tablas (relaciones), vistas, índices, ...

Término relacional	Término SQL
Tabla	Table
Fila	Row
Column	Column

Crear: `CREATE`
Eliminar: `DROP`
Modificar: `ALTER`

4.2. Definición de datos en SQL (2)

4.2.1. Esquemas y catálogos

Esquemas

* Permiten definir conjuntos de tablas y otros elementos
* Disponible desde SQL2
* Identificables mediante un nombre
* La creación del esquema se puede hacer en un solo paso (especificando sus elementos). También se puede declarar en esquema y luego incorporarle sus elementos

Ejemplo

```
CREATE SCHEMA BANCO AUTHORIZATION MLOPEZ
```

Catálogo: Conjunto de esquemas

4.2. Definición de datos en SQL (3)

4.2.2. Creación de tablas, tipos de datos y restricciones

Creación de tablas: `CREATE TABLE`

Ejemplo

```
i) CREATE TABLE BANCO.CLIENTE
ii)CREATE TABLE CLIENTE
```

Definición de una tabla

- Nombre de la tabla
- Declaración de atributos: Nombre, dominio, restricciones
- Restricciones: Clave, integridad referencial, y demás

4.2. Definición de datos en SQL (4)

4.2.2. Creación de tablas, tipos de datos y restricciones

Ejemplo de creación de tablas

```
CREATE TABLE CLIENTES
    (NOMBRECLI VARCHAR(50) NOT NULL,
    DNICLI VARCHAR(8) NOT NULL,
    DOMICILIO VARCHAR(50) NOT NULL,
    PRIMARY KEY (DNICLI)
    ON DELETE CASCADE ON UPDATE CASCADE
    );
CREATE TABLE CUENTAS
    (NUMEROCTA VARCHAR(10) NOT NULL,
    SALDO DECIMAL(12,2)  NOT NULL,
    NOMBRESUC VARCHAR(50) NOT NULL,
    PRIMARY KEY (NUMEROCTA)
    ON DELETE CASCADE ON UPDATE CASCADE
    );
```

Atributos

Restricciones

Atributos

Restricciones

4.2. Definición de datos en SQL (5)

4.2.2. Creación de tablas, tipos de datos y restricciones

Ejemplo de creación de tablas

```
CREATE TABLE CTACLI
    (DNICLI VARCHAR(8) NOT NULL,
    NUMEROCTA VARCHAR(20) NOT NULL,
    PRIMARY KEY (DNICLI, CTACLI),
    FOREIGN KEY (DNICLI) REFERENCES
     CLIENTES(DNICLI),
    FOREIGN KEY (NUMEROCTA) REFERENCES
     CUENTAS(NUMEROCTA)
    );
```

Atributos

Restricciones

4.2. Definición de datos en SQL (6)

4.2.3. Tipos de datos y dominios en SQL

Caracteres, numéricos, fecha y hora

* Enteros: `INTEGER` o `INT`, `SMALLINT`
* Reales: `FLOAT`, `REAL`, `DOUBLE PRECISION`
* Cadena de longitud fija: `CHAR(n)` o `CHARACTER(n)`
* Cadena de longitud variable: `VARCHAR(n)`
* Fecha: `DATE` Componentes: `YEAR`, `MONTH`, `DAY`
  Formato: YYYY-MM-DD
* Hora: `TIME` Componentes: `HOUR`, `MINUTE`, `SECOND`
  Formato: HH:MM:SS.

4.2. Definición de datos en SQL (7)

4.2.4. Definición de claves primarias y externas. Especificación de restricciones

Definición de clave: `PRIMARY KEY`

Los atributos que forman la clave deben ser `NOT NULL`

Definición de clave externa: `FOREIGN KEY ... REFERENCES`

```
CREATE TABLE CTACLI
       (DNICLI VARCHAR(8) NOT NULL,
       NUMEROCTA VARCHAR(20) NOT NULL,
       PRIMARY KEY (DNICLI, CTACLI),
       FOREIGN KEY (DNICLI) REFERENCES
    CLIENTES(DNICLI),
       FOREIGN KEY (NUMEROCTA) REFERENCES
    CUENTAS(NUMEROCTA)
       );
```


4.2. Definición de datos en SQL (8)

4.2.5. Modificación de tablas

Modificación de tablas: `ALTER TABLE`

* Añadir o eliminar columnas
* Modificar la definición de una columna
* Añadir o eliminar restricciones de columna

Ejemplo (Añadir columna):

```
ALTER TABLE BANCO.CLIENTES ADD CIUDAD
VARCHAR(20);
```

Ejemplo (Eliminar columna):

i) Propagación en cascada (`CASCADE`)

ii) No eliminar si hay relacionados (`RESTRICT`)

```
ALTER TABLE BANCO.CLIENTES DROP CIUDAD
CASCADE;
```

4.2. Definición de datos en SQL (9)

4.2.6. Eliminación de tablas y esquemas

* Eliminación de tablas: `DROP TABLE`
* Eliminación de esquemas: `DROP SCHEMA`

Si hay elementos relacionados se puede detener el proceso (`RESTRICT`) o se puede propagar el efecto (`CASCADE`)

Ejemplo:

```
DROP TABLE CLIENTES RESTRICT;
DROP SCHEMA BANCO CASCADE;
```


4.3. Estructura básica de una consulta SQL (1)

Componentes básicos de una sentencia SQL

* Cláusula `SELECT`: Proyección. Atributos deseados
* Cláusula `FROM`: Producto cartesiano. Tablas involucradas
* Cláusula `WHERE`: Selección. Condiciones o predicados

Sentencia SQL típica

```
SELECT A1, A2, ..., An
FROM R1, R2, ..., Rm
WHERE P
```

equivalente a

$\Pi_{A1, A2, ..., An}(\sigma_P (R_1 \times R_2 \times ... \times R_m))$

El uso de `SELECT *` devuelve todos los atributos

4.3. Estructura básica de una consulta SQL (2)

Ejemplo: Nombres de todos los clientes

```
SELECT nombreCli
FROM clientes;
```

Ejemplo: Nombres de los empleados de Downtown

```
SELECT nombreEmp
FROM empleados
WHERE nombreSuc = "Downtown";
```

Eliminación de duplicados

```
SELECT DISTINCT
```

Ejemplo: Nombres diferentes de empleados de Downtown

```
SELECT DISTINCT nombreEmp
FROM empleados
WHERE nombreSuc = "Downtown";
```


4.3. Estructura básica de una consulta SQL (3)

Conservación de duplicados (Predeterminado)

```
SELECT ALL
```

Uso de operadores aritméticos

```
SELECT numeroCta, saldo * 166.386
FROM cuentas;
```

4.4. Predicados y conectores (1)

Uso en la especificación de predicados implícitos (para combinar tablas) y de predicados explícitos.

```
SELECT empleados.nombreEmp
FROM sucursales, empleados
WHERE sucursales.nombreSuc = empleados.nombreSuc;
```

Conectores lógicos: `AND`, `OR` y `NOT`

Ejemplo: Nombres de empleados de la ciudad de Brooklyn

```
SELECT empleados.nombreEmp
FROM sucursales, empleados
WHERE sucursales.nombreSuc = empleados.nombreSuc
      AND sucursales.ciudadSuc = "Brooklyn";
```


4.4. Predicados y conectores (2)

Operador `BETWEEN`

Ejemplo: Números de cuenta con saldos comprendidos entre 20000 y 50000

```
SELECT numeroCta
FROM cuentas
WHERE saldo BETWEEN 20000 AND 50000;
```

Comparación aproximada: Operador `LIKE`

Uso de comodines: 1 cárácter _
Varios caracteres %

Ejemplo: Nombres de clientes que vivan en cualquier casa de la calle Gibraltar español

```
SELECT nombreCli
FROM clientes
WHERE domicilio LIKE "Gibraltar español%";
```

4.5. Ordenación de tuplas

Ordenación mediante ORDER BY
ORDER BY va después del WHERE, si procede.
Ordenación ascendente de forma predeterminada
Ejemplo: Lista alfabética de clientes con cuenta en Downtown

```
SELECT DISTINCT nombreCli
FROM clientes, cuentas, ctacli
WHERE clientes.dniCli = ctacli.dniCli AND
cuentas.numeroCta = ctacli.numeroCta AND
     cuentas.nombreSuc = "Downtown"
ORDER BY nombreCli;
```

Ordenación descendente: Uso de `DESC` después del nombre de columna.

4.6. Creación de alias (1)

Variables de tupla: Permiten la comparación de dos filas de la misma tabla. Renombra a una tabla
Ejemplo: Empleados que trabajan con Smith

```
SELECT E.nombreEmp
FROM Empleados E, Empleados Ebis
WHERE Ebis.nombreEmp = "Smith" AND
Ebis.nombreSuc = E.nombreSuc;
```

Ejemplo: Sucursales con activo superior al de alguna sucursal de Horseneck

```
SELECT E.nombreEmp
FROM Sucursales S1, Sucursales S2
WHERE S1.ciudadSuc = “Horseneck” AND
S2.Activo > S1.Activo;
```

4.6. Creación de alias (2)

Para renombrar atributos se usa `AS`

```
SELECT nombreSuc AS nombreDeSucursales
FROM sucursales;
```

Bastate útil cuando se tienen nombres “complicados” para una columna” o se han realizado operaciones.

Ejemplo:

```
SELECT numeroCta, saldo * 166.386 AS SaldoAPesetas
FROM cuentas;
```


4.7. Operaciones de conjuntos (1)

Unión (`UNION`), Intersección (`INTERSECT`) y Diferencia (`MINUS`)

Ejemplo: Nombre de todas las personas

```
SELECT nombreEmp
FROM empleados
UNION
SELECT nombreCli
FROM clientes;
```

Ejemplo: Nombres de empleados que coinciden con clientes

```
SELECT nombreEmp
FROM empleados
INTERSECT
SELECT nombreCli
FROM clientes;
```

4.7. Operaciones de conjuntos (2)

Ejemplo: Nombres de empleados que no coinciden con clientes

```
SELECT nombreEmp
FROM empleados
MINUS
SELECT nombreCli
FROM clientes;
```

Estas 3 operaciones eliminan los duplicados de forma predeterminada. Para conservar los duplicados utilizaremos `UNION ALL`, `INTERSECT ALL`, y `MINUS ALL`

4.8. Consultas anidadas y pertenencia a conjuntos (1)

Para realizar una comparación con un conjunto de valores podemos utilizar conjuntos explícitos en lugar de varios OR

Ejemplo: Empleados que trabajan en Brighton o Downtown

```
SELECT nombreEmp
FROM empleados
WHERE nombreSuc IN ("Downtown", "Brighton");
```

También podemos utilizar `NOT IN` para la no pertenencia

Otro uso es que los valores se obtengan como resultado de una consulta -> Consultas anidadas

4.8. Consultas anidadas y pertenencia a conjuntos (2)

Ejemplo: Empleados que trabajan en sucursales de la ciudad de Brooklyn

```
SELECT nombreEmp
FROM empleados
WHERE nombreSuc IN (SELECT nombreSuc
                    FROM sucursales
                    WHERE ciudadSuc = "Brooklyn");
```

Si hay ambigüedades en los nombres de las columnas se solucionan buscando en la consulta más interna

4.8. Consultas anidadas y pertenencia a conjuntos (3)

4.8.1. Consultas correlacionadas

Son un tipo especial de consultas anidadas

Son consultas en las que la consulta anidada se ejecuta una vez para cada fila de la consulta externa

Se consigue cuando en el WHERE de la subconsulta se hace referencia a columnas de la consulta externa

Nombre	Apellidos	NSS	AñoNacimiento	Sexo	Salario	NSSSuperv
Pedro	Márquez	1	1954	H	1200	2
Isabel	Fernández	2	1972	M	1150	3
María	Yagüe	3	1963	M	2200	null
Juan	Martín	4	1971	H	1050	3

Personal

Nombre	NSSP	AñoNacimiento	Sexo	Parentesco
Juana	1	1974	M	Hija
Manuel	1	1976	H	Hijo
Margarita	1	1958	M	Esposa
María	3	1993	M	Hija
Luis	3	1963	H	Esposo

Familia

4.8. Consultas anidadas y pertenencia a conjuntos (4)

4.8.1. Consultas correlacionadas

Ejemplo: Nombre y apellidos de trabajadores con el mismo nombre y sexo que sus familiares

```
SELECT Nombre, Apellidos
FROM Personal
WHERE Nombre IN
      (SELECT Nombre
      FROM Familia
      WHERE NSS=NSSP AND
Personal.Nombre = Familia.Nombre AND
Personal.Sexo = Familia.Sexo);
```

También se podría hacer combinándolas con el NSS

4.8. Consultas anidadas y pertenencia a conjuntos (5)

4.8.2. Función `EXISTS`

Permite comprobar si el resultado de una consulta es vacío

Ejemplo: Nombre y apellidos de trabajadores para los que existan familiares con su nombre y su sexo

```
SELECT Nombre, Apellidos
FROM Personal
WHERE EXISTS
      (SELECT *
      FROM Familia
      WHERE NSS=NSSP AND
Personal.Nombre = Familia.Nombre AND
Personal.Sexo = Familia.Sexo);
```

También se puede utilizar `NOT EXISTS`

4.9. Comparación de conjuntos

Se usa cuando el predicado supone la comparación con los algunos o todos los elementos de un conjunto

Para ello se utiliza `SOME` y `ALL`

Ejemplo: Sucursales que tienen un activo superior a cualquiera de las sucursales de la ciudad de Horseneck

```
SELECT nombreSuc
FROM sucursales
WHERE activo > SOME (SELECT activo
                     FROM sucursales
                     WHERE ciudadSuc = "Horseneck");
```

Para comparar con todos los valores se utiliza `ALL`

Ambos se puede combinar con los operadores de comparación, dando lugar a

`< SOME, <= SOME, >= SOME, <> SOME, = SOME, < ALL, <= ALL, >= ALL, = ALL y <> ALL`

4.10. Funciones de agregación (1)

Son funciones aplicadas a grupos de filas

- Promedio: `AVG`
- Mínimo: `MIN`
- Máximo: `MAX`
- Total: `SUM`
- Cuenta: `COUNT`

Los grupos se establecen mediante `GROUP BY`

La cláusula `GROUP BY` va después de la cláusula `WHERE`

Las filas que se agrupan tienen el mismo valor en las columnas por las que se realiza la agrupación

4.10. Funciones de agregación (2)

Ejemplo: Activo medio por ciudades escribiríamos lo siguiente:

```
SELECT ciudadSuc, AVG(activo)
FROM sucursales
GROUP BY ciudadSuc;
```

Es posible establecer condiciones sobre los grupos. Esto se hace mediante la cláusula `HAVING`

Ejemplo: Activos de las sucursales por ciudades para aquellas en que la media sea superior a 2000000, escribiríamos

```
SELECT ciudadSuc, AVG(activo)
FROM sucursales
GROUP BY ciudadSuc
HAVING AVG(activo) > 2000000;
```


4.10. Funciones de agregación (3)

Ejemplo: Sucursales que tienen más de una cuenta, y luego utilizar esto para obtener en qué ciudad está

```
SELECT ciudadSuc
FROM Sucursales
WHERE nombreSuc IN
(SELECT nombreSuc
FROM Cuentas
GROUP BY nombreSuc
HAVING COUNT(numeroCta) > 1);
```

Ejemplo: Número de ciudades en los que el banco tiene sucursales

```
SELECT COUNT (DISTINCT ciudadSuc)
FROM Sucursales
```

4.11. Vistas (1)

Permiten la personalización de la información

Tablas derivadas a partir de otras (pueden ser vistas a su vez)

Son tablas virtuales: No tienen por que estar almacenadas físicamente

Presentan algunos problemas en la actualización

Definidas con `CREATE VIEW`

Ejemplo: Vista que contiene el nombre, teléfono y la ciudad en la que trabajan cada uno de los empleados

```
CREATE VIEW EmpleCiudad
AS SELECT nombreEmp, telefono, ciudadSuc
      FROM Empleados, Sucursales
      WHERE Empleados.nombreSuc = Sucursales.nombreSuc;
```


4.11. Vistas (2)

Vistas con especificación del nombre de las columnas

Ejemplo: Vista con el DNI de cada cliente y el saldo total de sus cuentas

```
CREATE VIEW ClientesSumaSaldo (DNICli, SaldoTotal)
AS SELECT DNICli, SUM(Saldo)
      FROM Clientes, CtaCli, Cuentas
      WHERE Clientes.DNICli = CtaCli.DNICli and
            CtaCli.numeroCta = Cuentas.numeroCta
      GROUP BY Clientes.DNICli;
```

Eliminación de vistas con `DROP VIEW`

Ejemplo: Eliminación de la vista ClientesSumaSaldo

```
DROP VIEW ClientesSumaSaldo;
```

4.12. Operaciones de modificación en SQL (1)

4.12.1. Inserción (INSERT)

Insertar un solo registro

Sintaxis:

```
INSERT INTO nombreTabla VALUES (valor1, valor2, ...);
```

Ejemplo:

```
INSERT INTO Empleados
VALUES ("Harry", "16", "161616", "Brighton");
```

Inserción de filas incompletas

```
INSERT INTO Empleados  (nombreEmp, dniEmp, nombreSuc)
VALUES ("Mukos", "17", "Brighton");
```

Insertar el resultado de una consulta

Sintaxis:

```
INSERT INTO nombreTabla ExpresionSELECT;
```


4.12. Operaciones de modificación en SQL (2)

4.12.2. Eliminación (DELETE)

Eliminar de una tabla las filas que cumplen una condición

Sintaxis:

```
DELETE
FROM Tabla
WHERE Condicion;
```

Ejemplo:

```
DELETE
FROM Empleados
WHERE nombreEmp = "Mukos";
```

4.12. Operaciones de modificación en SQL (3)

4.12.3. Actualización (`UPDATE`)

Actualizar en una tabla las filas que cumplen una condición

Sintaxis:

```
UPDATE tabla
SET Modificacion
WHERE Condicion
```

Ejemplo: Incrementar en un 10 por ciento el saldo de las cuentas de sucursales de la ciudad de Horseneck

```
UPDATE Cuentas
SET saldo = saldo * 1.1
WHERE nombreSuc IN (SELECT nombreSuc
                    FROM Sucursales
                    WHERE ciudadSuc = "Horseneck");
```


4.13. Otras formas de combinación de relaciones (1)

Hasta ahora, uso de `FROM` para indicar producto cartesiano

4.13.1. INNER JOIN

Forma compacta de combinar relaciones

Sintaxis:

```
Tabla1 INNER JOIN Tabla2 ON condicion
```

Ejemplo: Nombre de los empleados y la ciudad en la que trabajan

```
SELECT nombreEmp, ciudadSuc
FROM Empleados INNER JOIN Sucursales ON
Empleados.nombreSuc = Sucursales.nombreSuc;
```

4.13. Otras formas de combinación de relaciones (2)

4.13.1. INNER JOIN

Ejemplo: Clientes y saldos de cada una de sus cuentas

```
SELECT nombreCli, saldo
FROM Cuentas INNER JOIN
(Clientes INNER JOIN CtaCli ON
      Clientes.dniCli = CtaCli.dniCli) ON
Cuentas.numeroCta = CtaCli.numeroCta;
```


4.13. Otras formas de combinación de relaciones (3)

4.13.2. NATURAL INNER JOIN

Evita especificar la condición de *join*

Realiza la combinación basándose en columnas comunes

Ejemplo: Empleados y ciudad en la que trabajan

```
SELECT nombreEmp, ciudadSuc
FROM Empleados NATURAL INNER JOIN Sucursales;
```

4.13. Otras formas de combinación de relaciones (4)

4.13.3. Reuniones externas

Relajan la condición de *join* respecto al `INNER JOIN`
Permiten combinar registros de una tabla que no tengan registros relaciones en otra

•Reunión externa izquierda: `LEFT OUTER JOIN`
Permite registos de la tabla izquierda que no tengan registros relacionados en la tabla derecha

•Reunión externa derecha: `RIGHT OUTER JOIN`
Permite registos de la tabla derecha que no tengan registros relacionados en la tabla izquierda

•Reunión externa completa: `FULL OUTER JOIN`
Devuelve los registros de ambas tablas aunque no tengan registros relacionados

4.13. Otras formas de combinación de relaciones (5)

4.13.3. Reuniones externas

Ejemplo: Reunión externa izquierda de Sucursales de Horseneck con Empleados

```
SELECT Sucursales.nombreSuc, nombreEmp
FROM Sucursales LEFT OUTER JOIN Empleados ON
Sucursales.nombreSuc = Empleados.nombreSuc
WHERE ciudadSuc = "Horseneck";
```

Sucursales.nombreSuc	nombreEmp
Perrydge	Hansen
Perrydge	Dubitzky
Mianus	Henson
Round Hill	Null

Ejemplo: Reunión externa de Sucursales con Empleados

```
SELECT Sucursales.nombreSuc, nombreEmp
FROM Sucursales FULL OUTER JOIN Empleados ON
Sucursales.nombreSuc = Empleados.nombreSuc;
```