BASES DE DATOS

CLASE 6

SQL

- Structured Query Language (SQL): lenguaje de trabajo estándar para el modelo relacional
- Orígenes
  - AR / CRT → SEQUEL → SQL
    - 1986 → SQL1 (ANSI SQL)
    - 1992 → SQL2
    - 1999 → SQL2000
    - 2003 → SQL3 (SQL2003)
    - SQL2005, SQL2008, SQL2012, ...

SQL

- Es un estándar respetado por la mayoría de los productos
- Contiene herramientas para proveer integridad, seguridad, transacciones, vistas, entre otros
- Componentes principales
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)

SQL

- **DDL** → Lenguaje de definición de datos
  - Definición y modificación de:
    - Bases de datos
    - Esquemas
    - Relaciones
    - Índices
    - Vistas
  - Autorizaciones al **acceso a datos**
  - Definición de reglas de **integridad**
  - Control de **concurrencia**

SQL

- **DDL** → Lenguaje de definición de datos
  - Operaciones básicas:
    - **CREATE**
    - **DROP**
    - **ALTER**
  - Ejemplos
    - Creación/Modificación/Eliminación **BD**
    - Creación/Modificación/Eliminación **Tablas**
    - Creación/Eliminación **Índices**

SQL

- **DDL** → Lenguaje de definición de datos

CREATE DATABASE nombreBD;

DROP DATABASE nombreBD;

ALTER DATABASE nombreBD (**ALTER NAME** nuevoNombreBD)

SQL

- **DDL** → Lenguaje de definición de datos

```
CREATE TABLE empresa (
    idEmpresa INTEGER NOT NULL AUTO_INCREMENT,
    nombreEmpresa VARCHAR(100) NOT NULL,
    cuit VARCHAR(13) NOT NULL,
    direccion VARCHAR(50) NULL,
    ...,
    PRIMARY KEY(idEmpresa)
    );
```

SQL

- **DDL** → Lenguaje de definición de datos

```
CREATE TABLE empleado (
        idEmpleado INTEGER NOT NULL AUTO_INCREMENT,
        idEmpresa INTEGER UNSIGNED NOT NULL,
        fechaDesde DATE NOT NULL,
        fechaHasta DATE NULL,
        nombreEmpleado VARCHAR(100) NOT NULL,
        ...,
        PRIMARY KEY(idEmpleado)
        FOREIGN KEY(idEmpresa) REFERENCES empresa (idEmpresa)
                                        ON DELETE RESTRICT
                                        ON UPDATE NO ACTION );
```

- **DDL** → Lenguaje de definición de datos

```
DROP TABLE nombreTabla;

ALTER TABLE empresa (
    ADD COLUMN razonSocial VARCHAR(100) NOT NULL,
    DROP COLUMN nombreEmpresa,
    ALTER COLUMN direccion VARCHAR(80) NOT NULL,
    ...,
    );
```

- **DDL** → Lenguaje de definición de datos

CREATE UNIQUE INDEX indicePrimarioEmpresa

ON empresa **USING** HASH (idEmpresa)

CREATE INDEX indiceSecNombreEmpleado

ON empleado **USING** BTREE (nombreEmpleado)

DROP INDEX indiceSecNombreEmpleado;

UNIQUE: el sistema comprueba si existen valores duplicados en la tabla cuando se crea el índice (si ya existen datos) y cada vez que se añaden datos. Los intentos de insertar o actualizar datos duplicados generarán un error.

SQL

DML

- **DML** → Lenguaje de manipulación de datos
  - Consulta y modificación del contenido de la BD → los datos almacenados
  - Estructura básica → 3 cláusulas:
    - **SELECT**
    - **FROM**
    - **WHERE**

SQL

- **DML** → Lenguaje de manipulación de datos
  - SQL se basa en la teoría introducida por el álgebra relacional
  - π $a_1,\ldots,a_N$ (σ **P** (r_1 **x**...**x** r_M)) equivale a:

SELECT a_1 , ..., a_N
FROM r_1 , ..., r_M
WHERE **P**

SQL

- **SELECT**
  - La cláusula **SELECT** permite seleccionar los atributos que se desean mostrar en el resultado
  - * → Incluye todos los atributos de las tablas que aparecen en el **FROM**
  - **DISTINCT** → elimina tuplas duplicadas
  - **ALL** → permite todas las tuplas, aún las repetidas (valor por defecto)

SQL

DML

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- Consultas básicas
  - **Ej1**: datos de las sucursales almacenadas en la tabla sucursal
  - **Ej2**: id de las sucursales que dieron algún préstamo

SQL

DML

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- Es posible utilizar **operadores** en la cláusula **SELECT**
  - **Ej3**: obtener todos los nros de préstamo con sus importes incrementados en un 15%

SQL

DML

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- Es posible utilizar **operadores** en la cláusula **WHERE** (lógicos, BETWEEN-AND, etc.)
  - **Ej4**: nros de préstamo que se realizaron en la sucursal con id 37 y con monto superior a $200.000
  - **Ej5**: nros de préstamo cuyo monto está entre $20.000 y $100.000

SQL

DML

- **FROM**
  - La cláusula **FROM** permite realizar un producto cartesiano entre las tablas que se incluyan
  - En la cláusula **WHERE** se debe especificar las condiciones de cruce entre las tablas
- **AS**
  - La cláusula **AS** permite **renombrar** relaciones (tablas) o atributos
    - Es útil, por ejemplo, para realizar productos cartesianos de una tabla consigo misma

SQL

DML

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- **Ej6**: nombre de los clientes y el nro de prestamo correspondiente, de los prestamos de la sucursal con id 64.

- **Ej7**: id y nombre de las sucursales que poseen activo mayor que al menos una sucursal situada en “Buenos Aires”.

- **LIKE** (**%**, **_**)
  - La cláusula **LIKE** permite realizar distintas operaciones sobre **STRINGS**
    - **“Alfa%”**: cualquier cadena que empiece con “Alfa”
    - **“%casa%”**: cualquier cadena que tenga “casa” en su interior
    - **“_ _ _”**: cualquier cadena con tres caracteres
    - **“_ _ _%”**: cualquier cadena con al menos tres caracteres
    - **Ej8**: id y nombre de clientes que comiencen con la letra G y su domicilio contenga la cadena ‘Los Hornos’

- **ORDER BY**
  - La cláusula **ORDER BY** permite especificar los atributos por los cuáles serán **ordenadas** las tuplas del resultado
  - **DESC**, **ASC**: es posible indicar si el orden debe ser descendente o ascendente (por defecto)
    - **Ej9**: presentar todos los clientes ordenados por nombre
    - **Ej10**: presentar los nros de préstamo, nombre de sucursal e importe de todos los prestamos, ordenado por el nombre de sucursal (alfabéticamente) y por el importe (de mayor a menor)

SQL

- **UNION**, **INTERSECT**, **EXCEPT**
  - Permiten realizar **operaciones sobre conjuntos**. Los operandos involucrados (subconsultas) deben ser **compatibles**, es decir, deben tener la misma estructura y dominios
  - **UNION** agrupa las tuplas de dos subconsultas
    - **UNION ALL**: conserva tuplas duplicadas
  - **INTERSECT** intersecta las tuplas de dos subconsultas
  - **EXCEPT** realiza la diferencia entre dos subconsultas

SQL

DML

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- **Ej11**: id de clientes con cuentas o prestamos en cualquier sucursal
- **Ej12**: id de clientes con cuentas y prestamos en cualquier sucursal
- **Ej13**: id de clientes con cuentas y sin prestamos en cualquier sucursal

- **FUNCIONES DE AGREGACIÓN**
  - **AVG** → **promedio**: aplicable a atributos numéricos, retorna el promedio de la cuenta
  - **MIN** → **mínimo**: retorna el menor elemento no nulo dentro de las tuplas para ese atributo
  - **MAX** → **máximo**: retorna el mayor elemento no nulo dentro de las tuplas para ese atributo
  - **SUM** → **total**: aplicable a atributos numéricos, realiza la suma matemática
  - **COUNT** → **contador**: cuenta las tuplas resultantes

SQL

DML

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- **Ej14**: cantidad de cuentas con saldo mayor a $50000
- **Ej15**: saldo promedio de las cuentas de la sucursal ‘LP-1’
- **Ej16**: mayor importe otorgado en un préstamo de la sucursal ‘LP-2’
- **Ej17**: importe total asignado a prestamos en todas las sucursales

- **GROUP BY**
  - Permite **agrupar** un conjunto de tuplas por algún criterio
    - IMPORTANTE → los atributos proyectados en el resultado (que no usan funciones de agregación) deben ser usados como criterio de agrupación
  - Se aplica una **función de agregación** sobre cada grupo
  - Se puede además aplicar ciertas condiciones de grupo mediante la cláusula **HAVING** → actúa como **filtro**

SQL

DML

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- **Ej18**: saldo promedio de las cuentas de c/ sucursal
- **Ej19**: cantidad de clientes de cuentas de cada sucursal
- **Ej20**: saldo promedio de las cuentas de c/ sucursal, siempre y cuando supere los $200.000
- **Ej21**: id y nombre del cliente, y el saldo promedio de sus cuentas, siempre y cuando vivan en “La Plata” y tengan al menos 3 cuentas

SQL

DML

- **NULL**
  - La cláusula **IS NULL** / **IS NOT NULL** permite consultar por valores nulos → **no significa 0**
    - **Ej22**: mostrar aquellos préstamos con importe nulo
- **IN**
  - Permite consultar si un elemento es parte o no de un conjunto de tuplas
    - **Ej23**: id de clientes con cuentas y préstamos en cualquier sucursal (alternativa a ∩)
    - **Ej24**: id de clientes con cuentas y préstamos en alguna sucursal de “La Plata”

SQL

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- **SOME**, **ALL** → comparación de conjuntos
  - **Ej25**: presentar las sucursales que tengan activo mayor que alguna otra
  - **Ej26**: presentar la sucursal que tenga activo superior a todas
  - **Ej27**: encontrar la sucursal que tiene el mayor saldo promedio de sus cuentas

SQL

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- **EXIST** → devuelve verdadero si la subconsulta argumento no es vacía
  - **Ej28**: id de clientes con préstamos y cuentas en cualquier sucursal (alternativa a ∩)
  - **Ej29**: id de clientes que tienen cuentas en todas las sucursales de la ciudad de Buenos Aires

SQL

DML

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- **UNIQUE** → devuelve verdadero si la subconsulta argumento no produce tuplas duplicadas

  - **Ej30**: id de clientes que tienen una sola cuenta en la sucursal 'SL1'

SQL

VISTAS

- Las vistas son objetos que no contienen datos por si mismo
- Su contenido es el resultado de la ejecución de una consulta
- Se almacena sólo su definición
  - Se mantiene actualizada, ya que al utilizarse es **recalculada**
- Aplicaciones: modularización, seguridad, entre otras

SQL

VISTAS

Sucursal = (idSuc, nombreSuc, ciudadSuc, activo)
Cliente = (idCte, nombreCte, direccionCte, ciudadCte)
Prestamo = (idSuc, nroPrestamo, importe)
PropietarioPrestamo = (idCte, nroPrestamo)
Cuenta = (idSuc, nroCuenta, saldo)
PropietarioCuenta = (idCte, nroCuenta)

- **CREATE VIEW** nombreVista **AS** consultaAsociada

  - **Ej31**: crear una vista con los datos de los clientes "activos" de todas las sucursales. Luego utilizarla para consultar el nombre y dirección de los clientes activos de la sucursal 'CABA1'

```
CREATE VIEW clientesActivos AS
(  SELECT S.idSuc, nombreSuc, PP.idCte, nombreCte, direccionCte
   FROM Sucursal S, Prestamo P, PropietarioPrestamo PP, Cliente CL
   WHERE S.idSuc = P.idSuc AND P.nroPrestamo = PP.nroPrestamo
   AND PP.idCte = CL.idCte AND P.importe IS NOT NULL
  UNION
  ( SELECT S1.idSuc, nombreSuc, PC.idCte, nombreCte, direccionCte
    FROM Sucursal S1, Cuenta C, PropietarioCuenta PC, Cliente CL1
    WHERE S1.idSuc = C.idSuc
    AND C.nroCuenta = PC.nroCuenta  AND PC.idCte = CL1.idCte  AND C.saldo > 0) )
```

- **Ej31**: Utilizar la vista creada para consultar el nombre y dirección de los clientes activos de la sucursal ‘CABA1’

- **INSERCIÓN**
  - Para insertar una nueva tupla (fila) en una tabla se utiliza la cláusula **INSERT INTO - VALUES**:
    - **INSERT INTO** nombreTabla (<nombreAtributo>,...) **VALUES** (<valorAtributo>,...)
  - Se puede especificar la tupla completa o sólo algunos atributos (columnas)
    - En el último caso se debe indicar explícitamente los nombres de las columnas que se van a completar
    - **Ej32**: agregar una cuenta

- **ELIMINACIÓN**
  - Para eliminar una tupla (fila) en una tabla se utiliza la cláusula **DELETE-FROM-WHERE**
    - **DELETE FROM** nombreTabla **[WHERE** condición**]**
  - Si no se especifica una condición → se eliminan todas las tuplas de la tabla
    - **Ej33**: borrar las cuentas de la sucursal con id 43 y saldo entre $1000 y $5000

- **ACTUALIZACIÓN**
  - Para actualizar los datos de una tupla (fila) en una tabla se utiliza la cláusula **UPDATE-SET-WHERE**
    - **UPDATE** nombreTabla **SET** nombreAtributo = valorAtributo **[WHERE** condición**]**
  - Si no se especifica una condición → se actualizan todas las tuplas de la tabla
    - **Ej34**: modificar la dirección y ciudad del cliente con id 734.
    - **Ej35**: modificar el saldo de todas las cuentas, incrementándolo en un 5%

SQL

- Hasta ahora la única forma de realizar uniones de relaciones que se vió fue la del **producto cartesiano**
- De la misma forma que en el álgebra relacional, existe la alternativa de realizar un producto natural
- Los productos naturales especifican en la cláusula **FROM** tanto las **tablas** a usar como las **condiciones** de unión entre las mismas

- **INNER JOIN**
  - Es un producto natural básico, en el que se debe indicar la condición de unión entre las tablas
    - El atributo en común queda repetido
    - **Ej36**: inner join e/ Prestamo y PropietarioPrestamo

Prestamo			PropietarioPrestamo	
IdSuc	nroPrestamo	importe	idCte	nroPrestamo
1	123	100.000	18	123
3	321	50.000	28	321
17	456	250.000	38	456
24	654	35.000	48	654

- **LEFT (OUTER) JOIN**
  - Inicialmente se calcula el **inner join** y luego cada tupla perteneciente a la relación de la izquierda que no encontró par, aparece en el resultado con valores nulos en los atributos del otro lado
    - **Ej37**: left join e/ Prestamo y PropietarioPrestamo

Prestamo			PropietarioPrestamo	
IdSuc	nroPrestamo	importe	idCte	nroPrestamo
1	123	100.000	null	null
3	321	50.000	28	321
17	456	250.000	null	null
24	654	35.000	48	654

- **RIGHT (OUTER) JOIN**
  - Se calcula el **inner join** y luego cada tupla perteneciente a la relación de la derecha que no encontró par, aparece en el resultado con valores nulos en los atributos del otro lado
    - **Ej38**: right join e/ Prestamo y PropietarioPrestamo

Prestamo			PropietarioPrestamo	
IdSuc	nroPrestamo	importe	idCte	nroPrestamo
null	null	null	18	123
3	321	50.000	28	321
17	456	250.000	38	456
null	null	null	48	654

- **FULL (OUTER) JOIN**
  - Idem anteriores, pero aparecen todas las tuplas que no encontraron par, tanto de la relación de la izquierda como de la relación de la derecha
    - **Ej39**: full join e/ Prestamo y PropietarioPrestamo

Prestamo			PropietarioPrestamo	
IdSuc	nroPrestamo	importe	idCte	nroPrestamo
null	null	null	18	123
3	321	50.000	28	321
17	456	250.000	null	null
24	654	35.000	48	654

- **NATURAL JOIN**
  - Es análogo al **inner join** pero evita que el atributo común (por el que se hace la unión) aparezca dos veces
    - **Ej40**: natural join e/ Prestamo y PropietarioPrestamo

Prestamo			PropietarioPrestamo
IdSuc	nroPrestamo	importe	idCte
1	123	100.000	18
3	321	50.000	28
17	456	250.000	38
24	654	35.000	48

SQL

Ej. adicionales

Proveedores = (nroProv, nombreProv, situación, ciudad)

Partes = (nroParte, color, nombreParte, situación, ciudad)

Proyectos = (nroProy, nombreProy, ciudad)

RPPP = (nroProv, nroParte, nroProy, cantidad)

A. Obtener todos los proyectos de Córdoba
B. Nros de proveedores que suministren partes al proyecto llamado ‘Proyecto A1’, ordenado por proveedor
C. Envíos con cantidad entre 300 y 500
D. Nros de proveedores, nros de proyectos y nros de partes, para aquellas tuplas en donde los tres elementos sean de la misma ciudad
E. Nombre y nros de partes suministradas por un proveedor de Córdoba a un proyecto de Córdoba
F. Cant. de proyectos que tenga a “Proveedor S1” como proveedor
G. Cant. total de partes “Parte AH3” suministradas por el proveedor “Proveedor S1”

SQL

Ej. adicionales

Proveedores = (nroProv, nombreProv, situación, ciudad)

Partes = (nroParte, color, nombreParte, situación, ciudad)

Proyectos = (nroProy, nombreProy, ciudad)

RPPP = (nroProv, nroParte, nroProy, cantidad)

H. Envíos que no tengan la cantidad nula
I. Obtener los colores de las partes suministras por el proveedor llamado “Proveedor S1”.
J. Obtener el nro y nombre de los proyectos para los cuales “Proveedor S1” es el único proveedor
K. Cambiar a ‘GRIS’ el color de las partes de color ‘ROJO’
L. Eliminar los proyectos que no tengan suministros.
M. Proveedores que vivan en la misma ciudad que el proveedor llamado “Proveedor S1”.

Ej. adicionales

Proveedores = (nroProv, nombreProv, situación, ciudad)

Partes = (nroParte, color, nombreParte, situación, ciudad)

Proyectos = (nroProy, nombreProy, ciudad)

RPPP = (nroProv, nroParte, nroProy, cantidad)

O. Nombres de los proveedores que suministran la parte P2.
P. Presentar aquellos proveedores que suministren todas las partes existentes en la tabla
R. Obtener los número de partes provistas por más de un proveedor. Para este caso considerar que un proveedor sólo participa en un proyecto.
S. Informar el número de parte que se suministre a un proyecto cualquiera tal que en promedio ese suministro supere 200.
T. Listar todos los proveedores existentes, indicando los nombres de los proyectos a los que el proveedor le suministra alguna parte.