Universidad Nacional de la Patagonia San Juan Bosco
Facultad de Ingeniería

Cátedra: **Bases de datos I**

QbyE & SQL

CUENTA	**nombre_sucursal**	**número_cuenta**	**saldo**
	Norte	*_x*	*_y*

CLI/CUEN	**id_cliente**	**número_cuenta**
	_y	*_x*

RESULTADO	**número_cuenta**	**id_cliente**	**saldo**
P.	*_x*	*_y*	*_y*

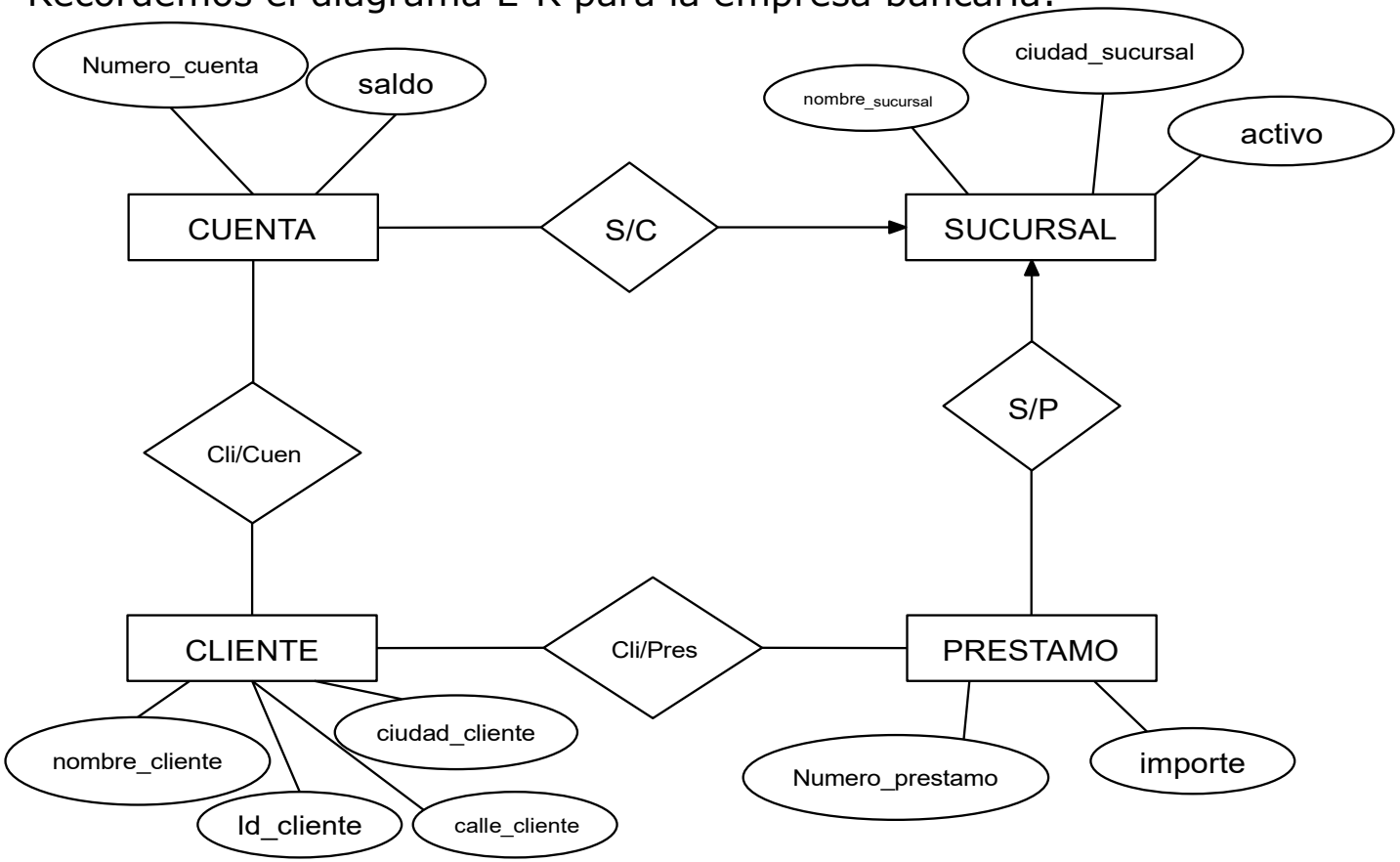
```
SELECT <lista atributos>
FROM <lista relaciones>
WHERE <condición>
```

- Query by Example. QBE
  - Se refiere a una familia de lenguajes que implementan las ideas del cálculo relacional de dominios. Es el nombre tanto de un lenguaje de manipulación de datos como el de un sistema de BD que incluyó a este lenguaje. El sistema QBE se desarrolló en el Centro de desarrollo T.J. Watson, de IBM, a principios de los años '70 y el lenguaje de manipulación de datos QBE se usó más tarde en QMF (Query Management Facility, mecanismo de gestión de consultas) como opción de interfaz para DB2.

- Características:
  - Sintaxis bidimensional → Las consultas se presentan en forma de tablas.
  - Una consulta en un lenguaje unidimensional (SQL) se puede formular en una línea. Un lenguaje bidimensional necesita dos dimensiones para la formulación de consultas.

  - Las consultas en QBE se expresan **mediante un ejemplo**:
  - ✓ **Esqueletos de tablas**: el esquema de la relación que se rellena con filas ejemplo.
  - ✓ **Fila ejemplo**: formada por constantes y elementos de ejemplo, son variables de dominio.
  - ✓ **Variables de dominio**: van precedidas por un guión bajo: _x
  - ✓ **Constantes**: aparecen sin ninguna indicación particular.
  - ✓ **Condiciones de selección sobre tuplas**:
    - **Conjunción**: por filas.
    - **Disyunción**: por columnas.

– Recordemos el diagrama E-R para la empresa bancaria:



– Los esquemas resultantes son:
  - ✓ Esquema_sucursal=(nombre_sucursal, activo, ciudad_sucursal)
  - ✓ Esquema_cliente=(id_cliente, nombre_cliente, calle_cliente, ciudad_cliente)
  - ✓ Esquema_cuenta=(número_cuenta, nombre_sucursal, saldo)
  - ✓ Esquema_préstamo=(número_préstamo, nombre_sucursal, importe)
  - ✓ Esquema_cli/cuen=(id_cliente, número_cuenta)
  - ✓ Esquema_cli/Pres= (id_cliente, numero_prestamo)

– Las relaciones quedan de esta manera:

CLIENTE

id_cliente	**nombre_cliente**	**calle_cliente**	**ciudad_cliente**
C-123	Lopez	San Martin	CRD
C-230	Sosa	Rivadavia	TRW
C-212	Herrera	Alem	MDY
C-235	Torres	Canada	CRD
C-189	Williams	Gales	TRW

SUCURSAL

nombre_sucursal	**ciudad_sucursal**	**activo**
Norte	CRD	1.800.000
Sur	TRW	420.000
Oeste	MDY	80.000
Centro	CRD	12.500.000
Roca	CRD	10.000.000

CUENTA

número_cuenta	nombre_sucursal	saldo
101	Norte	500
215	Sur	700
102	Oeste	400
305	Centro	350
201	Oeste	900
222	Norte	1800

PRESTAMO

nombre_sucursal	**nro_préstamo**	**importe**
Centro	*P-170*	*600*
Sur	*P-230*	*800*
Norte	*P-260*	*340*

CLI/CUE

id_cliente	**número_cuenta**
C-123	101
C-230	215
C-212	102
C-235	305
C-123	222

CLI/PRES

id_cliente	**número_préstamo**
C-212	17
C-230	23
C-189	15
C-235	14
C-230	93

– Siguiendo el ejemplo bancario, obtenga todos los números de préstamo de la sucursal Norte

PRÉSTAMO	nombre_sucursal	número_préstamo	importe
	Norte	*P._x*	

P . → Print

– QBE asume que una fila vacía contiene una variable única → si una variable no aparece más de una vez en una consulta, se puede omitir:

PRÉSTAMO	nombre_sucursal	número_préstamo	importe
	Norte	*P.*	

– QBE realiza **eliminación automática de duplicados**. Para evitarlo se inserta la orden **ALL.** después de P.

PRÉSTAMO	nombre_sucursal	número_préstamo	importe
	Norte	*P.ALL.*	

– Para mostrar la relación PRÉSTAMO completa, se crea una única fila con la orden P. en todos sus campos o bien, debajo del nombre de la relación:

PRÉSTAMO	nombre_sucursal	número_préstamo	importe
P.			

- QBE también permite formular consultas donde intervengan comparaciones aritméticas, P. e. encontrar todos los préstamos con una cantidad mayor a $4000

PRÉSTAMO	nombre_sucursal	número_préstamo	importe
		P.	*>4000*

- Las condiciones sólo pueden tener una expresión aritmética en el lado derecho de la operación, donde involucre tanto variables de dominio como constantes. El lado izquierdo del operador de comparación debe estar vacío.

P. e. > *(_x + _y -20)*

- Las operaciones aritméticas compatibles con QBE son: =, <, >, ¬, ≤ y ≥

- Considere la siguiente consulta: Obtener todos los préstamos de las sucursales cuyo nombre no sea *Sur*.
- La función principal de las variables es obligar a ciertas tuplas a tener el mismo valor en algunos atributos, considere la consulta: Obtener todos los préstamos de la sucursal Norte y Sur

PRÉSTAMO	nombre_sucursal	número_préstamo	importe
	Norte	*P._x*	
	Sur	*_x*	

- Considere ahora obtener los préstamos de la sucursal Norte o Sur

PRÉSTAMO	nombre_sucursal	número_préstamo	importe
	Norte	*P._x*	
	Sur	*P._y*	

- Consideremos ahora la consulta obtener los ID de los clientes que viven en la misma ciudad que *Gómez*

CLIENTE	id_cliente	nombre_cliente	calle	ciudad_cliente
		P._x		*_y*
		Gómez		*_y*

- Consultas sobre varias relaciones
  - QBE permite formular consultas que involucren varias relaciones, similar al producto cartesiano o join.
  - Las conexiones entre varias relaciones se hacen con variables, que obligan a algunas tuplas a tomar el mismo valor en ciertos atributos.
  - Supóngase que se desea encontrar los id's de todos los clientes que tienen un préstamo en la sucursal *Norte*.

PRÉSTAMO	nombre_sucursal	número_préstamo	importe
	Norte	*_x*	

CLI/PRES	id_cliente	número_préstamo
	P._y	*_x*

  - De manera similar, se podría encontrar todos los clientes que tengan una cuenta y un préstamo en el banco

CLI/CUEN	id_cliente	número_cuenta
	_x	

CLI/PRES	id_cliente	número_préstamo
	P._x	

- Negación
  - ✓ Bajo el nombre de una relación
  - – Consideremos la consulta: Obtener los id de todos los clientes que tienen una cuenta en el banco pero que no tienen un préstamo en el mismo
  - – En QBE las consultas que expresan negación se formulan con un símbolo not (¬) debajo del nombre de la relación y sobre una fila . P.e.:

CLI/CUEN	**id_cliente**	**número_cuenta**
	_x	

CLI/PRES	**id_cliente**	**número_préstamo**
¬	*P._x*	

  - – En este caso, el símbolo ¬ se puede interpretar como **no existe**

  - ✓ Bajo el nombre de un atributo, se usa para expresar la condición de ≠. P. e. Encontrar todos los clientes que tienen al menos dos cuentas

CLI/CUEN	**id_cliente**	**número_cuenta**
	P._x	*_y*
	_x	¬ *_y*

  - – Esta consulta muestra todos los atributos id_cliente que aparecen en al menos dos tuplas y que tienen un valor distinto para número_cuenta

- Cuadro de condición
  - Cuando es poco conveniente o imposible expresar todas las restricciones de las variables de dominio dentro de esqueletos de tablas se usa un cuadro de condición, que permite expresar restricciones generales sobre cualquiera de las variables de dominio.
  - QBE permite que aparezcan expresiones lógicas en una caja de condición. Los operadores lógicos son las palabras **and** y **or**.
  - Sea la consulta: Encontrar los números de cuenta cuyo saldo esté entre $1500 y $3000

CUENTA	**nombre_sucursal**	**número_cuenta**	**saldo**
		P.	*_x*

Condiciones
_x >1500
_x < 3000

  - Obtener todas las sucursales con activos superiores al activo de al menos una sucursal con sede en CRD

SUCURSAL	**nombre_sucursal**	**ciudad_sucursal**	**activo**
	P._x		*_y*
		CRD	*_z*

Condición
_y > _z

– QBE permite la aparición de expresiones aritméticas complejas dentro de una caja de condición. Se puede formular la consulta: Encontrar todas las sucursales que tienen activos al menos dos veces mayor al activo de una de las sucursales con sede en CRD. Del mismo modo que se resolvió la consulta anterior:

SUCURSAL	**nombre_sucursal**	**ciudad_sucursal**	**activo**
	P._x		*_y*
		CRD	*_z*

Condición
*_y > _z * 2*

– Para obtener el nombre de sucursal cuyo activo esté entre $25000 y $40000 y los incluya, pero no $30000:

SUCURSAL	**nombre_sucursal**	**ciudad_sucursal**	**activo**
	P._x		*_y*

Condición
_y = (≥ 25000 ***and*** *≤ 40000* ***and*** *¬30000)*

– Para obtener todas las sucursales que tienen sede en CRD y TRW

SUCURSAL	**nombre_sucursal**	**ciudad_sucursal**	**activo**
	P._y	*_x*	

Condición
_x = (CRD ***or*** *TRW)*

- La relación resultado
  - Hasta ahora, los resultados que se muestran aparecen en un único esquema de relación. Si el resultado de una consulta contiene atributos de varios esquemas de relación, se necesita crear una relación resultado.
  - Si tenemos la consulta: Obtener el cliente, el número de cuenta y el saldo de todas las cuentas de la sucursal *Norte*.
  - Según el álgebra relacional:
    1. Reunión de las relaciones CLI/CUEN y CUENTA.
    2. Proyección sobre los atributos nombre-cliente, número-cuenta y saldo.
  - En QBE:
    1. Se crea un esqueleto de tabla, denominado RESULTADO, con los atributos id_cliente, número_cuenta y saldo. (Este nombre no debe existir previamente).
    2. Se escribe la consulta de la siguiente manera:

CUENTA	**nombre_sucursal**	**número_cuenta**	**saldo**
	Norte	*_x*	*_y*

CLI/CUEN	**id_cliente**	**número_cuenta**
	_z	*_x*

RESULTADO	**número_cuenta**	**id_cliente**	**saldo**
P.	*_x*	*_z*	*_y*

- Presentación ordenada de las tuplas
  - QBE ofrece al usuario el control sobre el orden en el cual se presentan las tuplas resultantes. Este control se expresa mediante el uso de las órdenes:

    ✓ AO: Ascending Order (ascendente)
    ✓ DO: Descending Order (descendente)

  - AO(n) → **n** es el orden de aplicación cuando se especifican varios campos.

  - Sea la consulta: Listar todos los números de cuenta de la sucursal *Norte* en orden alfabético ascendente, con sus respectivos saldos en orden descendente

CUENTA	**nombre_sucursal**	**número_cuenta**	**saldo**
	Norte	*P.AO(1)._x*	*P.DO(2)._y*

- Operaciones de agregación
  - QBE incluye los siguientes operadores de agregación:
  - ✓ AVG
  - ✓ MAX
  - ✓ MIN
  - ✓ SUM
  - ✓ CNT

  - A todos estos operadores **se les debe agregar el sufijo .ALL** que asegura que no se eliminan los duplicados. Así, para encontrar el saldo total de todas las cuentas de la sucursal *Norte*:

CUENTA	**nombre_sucursal**	**número_cuenta**	**saldo**
	Norte		*P.SUM.ALL.*

  - Supongamos ahora que queremos eliminar los duplicados, como todas las operaciones de agregación **deben terminar con .ALL, se añade el operador .UNQ** para especificar que se eliminan los duplicados. Entonces, para encontrar el número total de clientes que tienen una cuenta en el banco:

CLI/CUEN	**id_cliente**	**número_cuenta**
	P.CNT.UNQ.ALL.	

– QBE también ofrece la posibilidad de calcular funciones sobre grupos de tuplas utilizando el operador **G.** que es análogo al constructor **grup by** de SQL. Para calcular el saldo medio de cada sucursal, mostrando por sucursal en orden ascendente:

CUENTA	**nombre_sucursal**	**número_cuenta**	**saldo**
	P.AO.G.		*P.AVG.ALL._x*

– Para calcular el saldo medio de las cuentas de aquellas sucursales con un saldo de cuenta medio mayor que $24000, se agrega la siguiente condición

Condición
AVG.ALL._x > 24000

– Para obtener los datos de los clientes que tienen más de una cuenta en el banco:

CLIENTE	**id_cliente**	**nombre_cliente**	**calle**	**ciudad_cliente**
P.	*_x*			

CLI/CUEN	**id_cliente**	**número_cuenta**
	G._x	*CNT.ALL._y*

Condición
CNT.ALL._y > 1

- SQL 92
  - SQL ha sido adoptado como un estándar oficial en EEUU por el American National Standards Institute (ANSI), y como estándar internacional por la International Standards Organization (ISO)

  - Es un lenguaje de alto nivel, los programadores pueden evitar especificar muchos detalles de la manipulación de datos que serían necesarios en otros lenguajes.

  - Sus consultas se optimizan de manera adecuada produciendo una ejecución más eficiente. El resultado de una consulta SQL es una relación.

  - Lenguaje de bases de datos completo (no sólo de consulta)
  - ✓ Definición y Manipulación de Datos (LDD + LMD)
  - ✓ Definición y destrucción de Vistas (LDV)
  - ✓ Creación y destrucción de índices (aunque en SQL-92 ya no existen)
  - ✓ Incorporación de SQL dentro de código escrito con un Lenguaje de Programación de propósito general
  - Los proveedores de SGBDR comerciales implementan variaciones. Algunas incluyen características que no están estandarizadas (triggers /reglas activas incluidos en la versión SQL:1999)

- Cláusula SELECT
  - Corresponde a la **operación proyección** del AR. Se usa para listar los atributos deseados del resultado de una consulta
- Cláusula FROM
  - Corresponde a la **operación producto cartesiano** del AR. Lista las relaciones que deben ser analizadas en la evaluación de la expresión.
- Cláusula WHERE
  - Corresponde al **predicado selección** del AR. Es un predicado que engloba a los atributos de las relaciones que aparecen en la cláusula FROM.
  - Se usan las conectivas AND, OR y NOT cuando hay más de un átomo en el predicado. En cada átomo se usan los operadores de comparación =, <, >, ≠,≤ o ≥.
- Consultas básicas

  `SELECT <lista atributos>` ← atributos cuyos valores va a obtener la consulta

  `FROM <lista relaciones>` ← relaciones necesarias para realizar la consulta

  `WHERE <condición>` ← expresión booleana para identificar las filas

  ✓ Selecciona las tuplas de `<lista relaciones>` que satisfacen `<condición>` y proyecta el resultado sobre los atributos de `<lista atributos>`

- JOIN - Tipos y Condiciones

  - Define el producto cartesiano de las relaciones que aparecen en ella
  - SQL92 incluye JOIN NATURAL en la cláusula FROM

  - Considere la consulta: Seleccionar los nombres y apellidos de los docentes que dictan materias y su salario es mayor que $500

```
SELECT DISTINCT p.apellido, p.nombre
FROM Dicta AS d, Profesor AS p
WHERE p.legajo = d.legajo AND p.salario > 500;
```

  - De manera similar, se puede definir la consulta como sigue:

```
SELECT DISTINCT p.apellido, p.nombre
FROM Dicta AS d INNER JOIN Profesor AS p
  ON p.legajo = d.legajo
WHERE p.salario > 500;
```

- JOIN - Tipos y Condiciones (Cont.)

  - Las operaciones de reunión toman como **entrada dos relaciones** y devuelven como **resultado otra relación**.

  - Cada variante de las operaciones de Join en SQL92 está formado por:

  - Un **tipo**:
    - ✓ Inner join
    - ✓ Left outer join
    - ✓ Right outer join
    - ✓ Full outer join → combinación de los dos anteriores

  - Una **condición**:
    - ✓ Natural
    - ✓ On <predicado>
    - ✓ Using (A_1, A_2,…,A_n) → los atributos de Join no son comunes a ambas relaciones.

- JOIN - Tipos y Condiciones (Cont.)
- Ejemplo de Inner Join

  – Sean las relaciones Préstamo y Prestatario

nombre_sucursal	nro_préstamo	importe
Centro	*P-170*	*600*
Sur	*P-230*	*800*
Norte	*P-260*	*340*

nombre_cliente	nro_préstamo
Santos	*P-170*
Gómez	*P-230*

  – Se puede definir la consulta como sigue:

```
SELECT *
FROM Préstamo AS r INNER JOIN Prestatario AS p
  ON r.nro_préstamo = p.nro_préstamo;
```

  – Los atributos de la relación resultante son todos los de la relación del lado izquierdo del Join y todos los atributos de la relación del lado derecho. Note que nro_préstamo aparece dos veces:

nombre_sucursal	nro_préstamo	importe	nombre_cliente	nro_préstamo
Centro	*P-170*	*600*	*Santos*	*P-170*
Sur	*P-230*	*800*	*Gómez*	*P-230*

- JOIN - Tipos y Condiciones (Cont.)
- Ejemplo de Left Outer Join

  – Sean las relaciones Préstamo y Prestatario

nombre_sucursal	nro_préstamo	importe
Centro	*P-170*	*600*
Sur	*P-230*	*800*
Norte	*P-260*	*340*

nombre_cliente	nro_préstamo
Santos	*P-170*
Gómez	*P-230*

  – Se puede definir la consulta como sigue:

```
SELECT *
FROM Préstamo AS r LEFT OUTER JOIN Prestatario AS p
  ON r.nro_préstamo = p.nro_préstamo;
```

  – La unión externa por la izquierda se calcula del siguiente modo: Primero se calcula el resultado del Inner Join, luego, para cada tupla izquierda que no encaje con una derecha se rellenan con *Null*.

nombre_sucursal	nro_préstamo	importe	nombre_cliente	nro_préstamo
Centro	*P-170*	*600*	*Santos*	*P-170*
Sur	*P-230*	*800*	*Gómez*	*P-230*
Norte	*P-260*	*340*	*Null*	*Null*

- JOIN - Tipos y Condiciones (Cont.)
- Ejemplo de Natural Inner Join

  – Sean las relaciones Préstamo y Prestatario

nombre_sucursal	nro_préstamo	importe
Centro	*P-170*	*600*
Sur	*P-230*	*800*
Norte	*P-260*	*340*

nombre_cliente	nro_préstamo
Santos	*P-170*
Gómez	*P-230*

  – Se puede definir la consulta como sigue:

```
SELECT *
FROM Préstamo AS r NATURAL INNER JOIN Prestatario AS p;
```

  – Esta expresión calcula la reunión natural de dos relaciones, donde el único atributo común es nro_préstamo. El resultado es el siguiente:

nombre_sucursal	nro_préstamo	importe	nombre_cliente
Centro	*P-170*	*600*	*Santos*
Sur	*P-230*	*800*	*Gómez*

- DISTINCT - ALL
  - Los lenguajes formales de consulta están basados en la noción matemática de que una relación es un conjunto → nunca aparecen tuplas duplicadas en las relaciones.
  - SQL no elimina duplicados → consume tiempo. Si se quiere eliminarlos se debe usar la palabra clave **DISTINCT**:

```
SELECT DISTINCT nombre_cliente
FROM Cliente;
```

  - Es importante resaltar que SQL permite usar la palabra clave **ALL** para especificar explícitamente que no se eliminan los duplicados:

```
SELECT ALL nombre_cliente
FROM Cliente;
```

- Selección incondicional
  - Equivale a una condición TRUE para todas las filas

```
SELECT nombre_cliente, calle
FROM Cliente;

SELECT nombre_cliente, calle, numero_cuenta
FROM Cliente, Cuenta;
```

- Uso del símbolo *
  - Para obtener los valores de todos los atributos de una relación no es necesario listar todos los nombres tras la cláusula SELECT → Uso de **

```
SELECT *
FROM Sucursal
WHERE ciudad_sucursal='CRD';

SELECT *
FROM Sucursal, Cuenta
WHERE ciudad_sucursal='CRD';
```

- Operaciones aritméticas
  - La cláusula SELECT puede contener expresiones aritméticas que contengan los operadores +, -, * y / operando sobre constantes o atributos numéricos de tuplas.
  - Suponga la consulta: Listar los saldos de los clientes que viven en CRD y su saldo es menor a $10000, aplicarle un aumento del 10%

```
SELECT nombre_cliente, calle, 1.1*saldo
FROM Cliente, Cuenta
WHERE ciudad_sucursal='CRD' AND saldo < 10000;
```

  → el valor real de los saldos en la relación CLIENTE no cambia

  - SQL92 proporciona tipos de datos especiales, como varias formas del tipo *fecha* y permite varias funciones para operar sobre ellos
  - Considere el esquema:
  - DICTA (legajo, id_materia, salario, fecha_ingreso)
  - y la consulta: Listar los legajos de los profesores que comenzaron a dictar una materia en el año 2006:

```
SELECT legajo
FROM Dicta
WHERE año(fecha_ingreso) = '2006';
```

- Operador Between
  - Siguiendo con el esquema DICTA, considere la consulta: Encontrar los legajos de los docentes cuyos salarios se encuentran entre $300 y $1000.

```
SELECT legajo
FROM Dicta
WHERE salario ≥ 300 AND ≤ 1000;
```

  - Considere reformular la consulta de la siguiente manera:

```
SELECT legajo
FROM Dicta
WHERE salario BETWEEN 300 AND 1000;
```

- Calificación
  - En SQL los nombres de los atributos deben ser únicos en cada relación
  - Consulta que referencia a varios atributos de igual nombre, pero de relaciones distintas → Ambigüedad
  - Sean los esquemas:
  - ✓ PROFESOR(legajo, nombre, apellido, título, teléfono, dirección, ciudad)
  - ✓ DICTA(legajo, id_materia, salario, fecha_ingreso)
  - ✓ MATERIA(id_materia, descripción, nombre_dpto)
  - ✓ DEPARTAMENTO (nombre_dpto, numero_dpto, ubicación)
  - ✓ OFICINA_DEPTO (numero_dpto, oficina)

  - Se pide que liste el número de departamento, nombre y oficina de los departamentos de Bases de Datos y de Sistemas Operativos:

```
SELECT Departamento.numero_dpto, nombre_dpto, oficina
FROM Departamento, Oficina_depto
WHERE Departamento.nombre_dpto IN ('BD', 'SO')
AND Departamento.numero_dpto = Oficina_depto.numero_dpto;
```

- Renombramiento - AS
  - Puede utilizarse para acortar nombres de relaciones
  - Se puede ubicar tanto en la cláusula SELECT como en FROM
  - Su sintaxis es: `nombre_anterior AS nombre_nuevo`

```
SELECT d.numero_dpto, d.nombre_dpto, o.oficina
FROM Departamento AS d, Oficina_depto AS o
WHERE d.nombre_dpto IN ('BD', 'Investigación')
AND d.numero_dpto = o.numero_dpto;
```

  - En el caso de `FROM Departamento AS d` → el AS es opcional
  - Considere la consulta: Obtener los legajos de los docentes que cobren más que al menos un docente que dicta la materia M123:

```
SELECT d.legajo
FROM Dicta AS d, Dicta AS t
WHERE d.salario > t.salario AND t.id_materia = 'M123';
```

  - En el caso de renombrar en la cláusula SELECT → renombrar el atributo en la relación resultado:

```
SELECT m.id_materia AS identificador
FROM Materia AS m
WHERE departamento = 'Matemática';
```

- Ordenamiento
  - SQL permite presentar las tuplas resultado de una consulta de forma ordenada → Cláusula **ORDER BY**
  - Ordenación según valores de uno o varios atributos:
    - ✓ Ascendente **ASC** (por defecto)
    - ✓ Descendente **DESC**
  - Muy costoso → las tuplas no se ordenan en disco: se ven ordenadas
  - Sea la consulta: Seleccionar las materias del departamento Informática, ordenado de manera descendente por el nombre

```
SELECT id_materia, descripción
FROM Materia AS m
WHERE m.departamento = 'INFORMATICA'
ORDER BY descripción DESC;
```

  - Dada la consulta: Listar los campos de la relación DICTA donde los docentes tengan un salario menor a $1000. Ordenar por fecha de ingreso y a igual fecha por legajo

```
SELECT d.*
FROM Dicta AS d
WHERE d.salario < 1000
ORDER BY d.fecha_ingreso, d.legajo;
```

- Funciones de Agregación
  - Toman un conjunto de valores de entrada y producen un único valor como salida
  - ✓ AVG: Promedio
  - ✓ MIN: Valor Mínimo
  - ✓ MAX: Valor Máximo
  - ✓ SUM: Sumatoria, Total
  - ✓ COUNT: Cuenta
  - La entrada de MIN, MAX, SUM y AVG debe ser una colección de números
  - COUNT Cuenta el número de tuplas o valores en una consulta

  - Sea la consulta: Listar el sueldo total del docente cuyo legajo es 1230

```
SELECT SUM(d.salario)
FROM Dicta AS d
WHERE d.legajo = 1230;
```

  - Dada la consulta: Obtener la cantidad de materias por departamento:

```
SELECT m.departamento, COUNT(m.id_materia) AS cantidad
FROM Materia AS m
GROUP BY m.departamento;
```

- Hay casos en lo que para aplicar las funciones de agregación se deben eliminar los duplicados.
- P.e. Listar la cantidad de profesores que trabajan por departamento. Observar que los legajos se repiten si los docentes dictan varias materias por departamento:

```
SELECT m.departamento, COUNT(DISTINCT d. legajo)
FROM Materia AS m, Dicta AS d
WHERE m.id_materia = d.id_materia
GROUP BY m.departamento;
```

- Cláusula HAVING
  - Es una condición sobre una función de agregación
  - Sea la consulta: Listar los departamentos y la cantidad de materias que se dictan, si la cantidad es mayor que 10:

```
SELECT m.departamento, COUNT(m.id_materia)
FROM Materia AS m
GROUP BY m.departamento
HAVING COUNT(m.id_materia) > 10;
```

  - Primero se evalúa el predicado de la cláusula WHERE y luego el de HAVING. Las tuplas que satisfacen al WHERE se agrupan según el ORDER BY y la cláusula HAVING se aplica a cada grupo.

- Valores Nulos – NULL
  - SQL permite el uso de valores nulos para indicar la ausencia de información sobre el valor del atributo.
  - Operador IS NULL , IS NOT NULL
    - ✓ v IS NULL → TRUE si v es NULL
    - ✓ v IS NOT NULL → es TRUE si v es un valor no NULL
  - Sea la consulta: Listar aquellos docentes que no tengan teléfono:

```
SELECT legajo
FROM Profesor
WHERE telefono IS NULL;
```

  - El uso de un valor nulo en las operaciones aritméticas y de comparación causa varias complicaciones:
  - El resultado de una **expresión aritmética** (+, -, *, /) es nulo si cualquiera de los valores de entrada es nulo
  - El resultado de cualquier **comparación** que involucre un valor nulo se puede considerar falso. En concreto, SQL92 trata el resultado de este tipo de comparaciones como **nulo** (desconocido), el cual no es **cierto** ni **falso**. Además permite preguntar si el resultado de una comparación es desconocido.
  - La existencia de valores nulos también complica el procesamiento de los operadores de agregación. En general, todas las funciones excepto COUNT ignoran los valores nulos. COUNT de una colección vacía da 0.

- Operaciones sobre cadenas
  - SQL permite la comparación de cadenas de caracteres mediante el operador **LIKE**
  - Caracteres reservados (comodines) :
    - ✓ '%' para subcadenas
    - ✓ '_' para caracteres

  - Suponga la consulta: Liste los nombres y apellidos de los profesores cuya dirección esté en la calle San Martín

```
SELECT nombre, apellido, direccion
FROM Profesor
WHERE direccion LIKE '%San Martín%';
```

  - De la misma manera, SQL permite buscar discordancia utilizando el operador **NOT LIKE**

- Subconsultas anidadas
  - SQL proporciona un mecanismo para las subconsultas anidadas → Expresión SELECT-FROM-WHERE que se anida dentro de otra consulta.
  - Un uso común es llevar a cabo comprobaciones de:
  - ✓ Pertenencia a conjuntos
  - ✓ Comparación de conjuntos
  - ✓ Cardinalidad de conjuntos

- Pertenencia a conjuntos
  - SQL utiliza el cálculo relacional para las operaciones que permiten comprobar la pertenencia de una tupla a una relación:
  - IN → Comprueba la pertenencia a un conjunto
  - NOT IN → Comprueba la no pertenencia
  - Pueden ser conjuntos enumerados o subconsultas.

  - Sea la consulta: Obtener los legajos de los profesores que no dictan las materias Análisis I o Algebra:

```
SELECT legajo
FROM Dicta
WHERE descripción NOT IN ('Análisis I', 'Algebra');
```

- Comparación de conjuntos
  - Sea la consulta: Obtener los legajos de los profesores que ganan más que algún profesor que dicta la materia Análisis I:

```
SELECT d.legajo
FROM Materia AS m, Dicta AS d, Dicta AS t
WHERE m.id_materia = t.id_materia
  AND d.salario > t.salario
  AND m.descripción = 'Análisis I';

SELECT legajo
FROM Dicta
WHERE salario > SOME(
                SELECT salario
                FROM Materia AS m, Dicta AS d
                WHERE d.id_materia = m.id_materia
                AND m.descripción = 'Análisis I');
```

  - La palabra clave ANY es sinónimo de SOME.
  - En el caso de querer listar los legajos de todos los profesores → **ALL**
  - SQL permite las comparaciones <, >, ≤, ≥, = y ≠.
  - Se verifica que: **= SOME** es idéntico a **IN** y **≠ ALL** es lo mismo que **NOT IN**

- Cardinalidad de conjuntos
- Comprobación de relaciones vacías
  - SQL incluye la posibilidad de comprobar si una subconsulta no produce ninguna tupla como resultado
  - El constructor **EXIST** devuelve el valor True si la subconsulta argumento no es vacía
  - Utilizando el constructor **NOT EXIST** se puede comprobar la no existencia de tuplas en el resultado de una subconsulta

- Comprobación de tuplas duplicadas
  - Para comprobar si una subconsulta produce como resultado tuplas duplicadas SQL incluye el constructor **UNIQUE**
  - UNIQUE devuelve el valor True si la subconsulta que se le pasa como argumento no produce tuplas duplicadas
  - La existencia de tuplas duplicadas en una subconsulta se puede comprobar utilizando el constructor **NOT UNIQUE**

- Evaluación de consultas
  - En una consulta SQL hay un máximo de 6 cláusulas
  - Sólo son obligatorias SELECT y FROM

  - **Orden de especificación** de las cláusulas:
  - ✓ SELECT <lista atributos> atributos o funciones que se van a obtener
  - ✓ FROM <lista relaciones> relaciones necesarias
  - ✓ WHERE <condición para tuplas> condiciones para selección de tuplas
  - ✓ GROUP BY <lista atributos agrupación> especificación del agrupamiento de tuplas
  - ✓ HAVING <condición para grupos> condición para selección de grupos de tuplas
  - ✓ ORDER BY <lista atributos ordenación> orden de presentación del resultado

- **Orden de evaluación** de las cláusulas:

1. FROM (es decir, el JOIN de relaciones, si se especifica más de una)
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

- Diversas formas de especificar una misma consulta → Flexibilidad
- Ventajas e inconvenientes de esta flexibilidad:
- El usuario elige la técnica o enfoque más cómodo → Confusión del usuario: ¿qué técnica uso?
- Algunas técnicas son más eficientes que otras → El usuario debe determinar cuál
- En condiciones ideales:

✓ Usuario: se preocupa sólo de especificar la consulta correctamente

✓ SGBD: se ocupa de ejecutar la consulta de manera eficiente

- **Recomendación**: Consultas con mínimo anidamiento y mínimo ordenamiento implícito