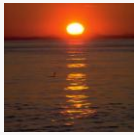
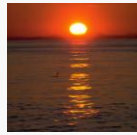
Capítulo 3: SQL

Database System Concepts, 5th Ed.



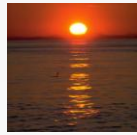
Capítulo 3: SQL

- Definición de datos
- Estructura básica de las consultas SQL
- Operaciones sobre conjuntos
- Funciones de agregación
- Valores nulos
- Subconsultas anidadas
- Consultas complejas
- Vistas
- Modificación de la BD
- Reunión de realciones



Historia

- IBM Sequel lenguaje desarrollado como parte del proyecto System R en el laboratorio de San Jose
- Renombrado Structured Query Language (SQL) Lenguaje estructurado de consultas
- ANSI y ISO publicaron una norma de SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999
  - SQL:2003
- Sistemas comerciales ofrecen casi todo, sino todo SQL-92 (características)
  - No todos los ejemplos funcionan en tu sistema (particular)



Lenguaje de Definición de Datos

Proporciona la especificación no solo de un conjunto de relaciones pero también la información de cada relación incluyendo:

- El esquema de cada relación
- El dominio de valores asociados con cada atributo
- Las restricciones de integridad
- El conjunto de índices que se debe mantener para cada relación.
- La información de seguridad y de autorización de cada relación.
- La estructura de almacenamiento físico de cada relación en el disco.

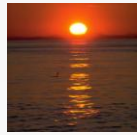


Figure 3.1: Esquema de la entidad bancaria

branch (*branch_name*, *branch_city*, *assets*)

sucursal (Nombre_sucursal, Ciudad_sucursal, activos)

customer (*customer_name*, *customer_street*, *customer_city*)

Cliente (Nombre_cliente, Calle_cliente, Ciudad_cliente)

loan (*loan_number*, *branch_name*, *amount*)

Prestamo(Numero_prestamo, Nombre_sucursal, Importe)

borrower (*customer_name*, *loan_number*)

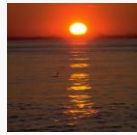
prestatario (Nombre_cliente, Numero_prestamo)

account (*account_number*, *branch_name*, *balance*)

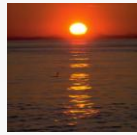
cuenta (Numero_cuenta, Nombre_sucursal, saldo)

depositor (*customer_name*, *account_number*)

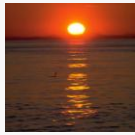
Impositor(Nombre_cliente, Numero_cuenta)



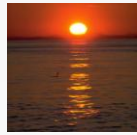
```
-- Create schema banco
--
CREATE DATABASE IF NOT EXISTS banco;
USE banco;
--
-- Definition of table `cliente`
--
DROP TABLE IF EXISTS `cliente`;
CREATE TABLE `cliente` (
  `Nombre_cliente` char(20) NOT NULL,
  `Calle_cliente` char(30) NOT NULL,
  `Ciudad_cliente` char(30) NOT NULL,
  PRIMARY KEY  (`Nombre_cliente`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```



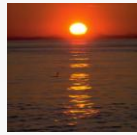
- INSERT INTO `cliente` (`Nombre_cliente`,`Calle_cliente`,`Ciudad_cliente`) VALUES
- ('Abril','Precialos','Valsaín'),
- ('Amo','Embajadores','Arganzuela'),
- ('Badorrey','Delicias','Valsaín'),
- ('Fernández','Jazmin','Leon'),
- ('Gómez','Carretas','Cerceda'),
- ('González','Arenal','La Granja'),
- ('López','Mayor','Peguerinos'),
- ('Pérez','Carretas','Cerceda'),
- ('Rodríguez','Yeserías','Cádiz'),
- ('Rupérez','Ramblas','León'),
- ('Santos','Mayor','Peguerinos'),
- ('Valdivieso','Goya','Vigo');



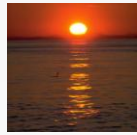
- DROP TABLE IF EXISTS `cuenta`;
- CREATE TABLE `cuenta` (
- `Numero_cuenta` char(10) NOT NULL,
- `Nombre_sucursal` char(15) NOT NULL,
- `Saldo` decimal(12,2) NOT NULL,
- PRIMARY KEY (`Numero_cuenta`),
- KEY `FK_Cuenta_1` (`Nombre_sucursal`),
- CONSTRAINT `FK_Cuenta_1` FOREIGN KEY (`Nombre_sucursal`) REFERENCES `sucursal` (`Nombre_sucursal`)
-) ENGINE=InnoDB DEFAULT CHARSET=latin1;



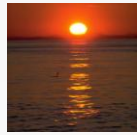
- NSERT INTO `cuenta` (`Numero_cuenta`,`Nombre_sucursal`,`Saldo`) VALUES
- ('c-101','Centro','500.00'),
- ('c-102','Navacerrada','400.00'),
- ('c-201','Galapagar','900.00'),
- ('c-215','Becerril','700.00'),
- ('c-217','Galapagar','750.00'),
- ('c-222','Moralzarzal','700.00'),
- ('c-305','Collado Mediano','350.00');



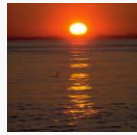
- DROP TABLE IF EXISTS `impositor`;
- CREATE TABLE `impositor` (
- `Nombre_cliente` char(20) NOT NULL,
- `Numero_cuenta` char(10) NOT NULL,
- PRIMARY KEY (`Nombre_cliente`,`Numero_cuenta`),
- KEY `FK_Impositor_2` (`Numero_cuenta`),
- CONSTRAINT `FK_Impositor_1` FOREIGN KEY (`Nombre_cliente`) REFERENCES `cliente` (`Nombre_cliente`),
- CONSTRAINT `FK_Impositor_2` FOREIGN KEY (`Numero_cuenta`) REFERENCES `cuenta` (`Numero_cuenta`)
-) ENGINE=InnoDB DEFAULT CHARSET=latin1;



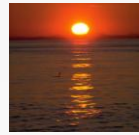
- NSERT INTO `impositor` (`Nombre_cliente`,`Numero_cuenta`) VALUES
- ('Gonzá¡lez','c-101'),
- ('López','c-102'),
- ('González','c-201'),
- ('Gómez','c-215'),
- ('Santos','c-217'),
- ('Rupérez','c-222'),
- ('Abril','c-305');



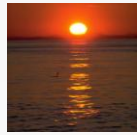
- DROP TABLE IF EXISTS `prestamo`;
- CREATE TABLE `prestamo` (
- `Numero_prestamo` char(10) NOT NULL,
- `Nombre_sucursal` char(15) NOT NULL,
- `Importe` decimal(12,2) NOT NULL,
- PRIMARY KEY (`Numero_prestamo`),
- KEY `FK_Prestamo_1` (`Nombre_sucursal`),
- CONSTRAINT `FK_Prestamo_1` FOREIGN KEY (`Nombre_sucursal`) REFERENCES `sucursal` (`Nombre_sucursal`)
-) ENGINE=InnoDB DEFAULT CHARSET=latin1;



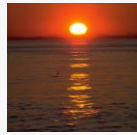
- INSERT INTO `prestamo` (`Numero_prestamo`,`Nombre_sucursal`,`Importe`) VALUES
- ('p-11','Collado Mediano','900.00'),
- ('p-14','Centro','1500.00'),
- ('p-15','Navacerrada','1500.00'),
- ('p-16','Navacerrada','1500.00'),
- ('p-17','Centro','1000.00'),
- ('p-23','Moralzarzal','2000.00'),
- ('p-93','Becerril','500.00');



- DROP TABLE IF EXISTS `prestatario`;
- CREATE TABLE `prestatario` (
- `Nombre_cliente` char(20) NOT NULL,
- `Numero_prestamo` char(10) NOT NULL,
- PRIMARY KEY (`Nombre_cliente`,`Numero_prestamo`),
- KEY `FK_Prestatario_2` (`Numero_prestamo`),
- CONSTRAINT `FK_Prestatario_1` FOREIGN KEY (`Nombre_cliente`) REFERENCES `cliente` (`Nombre_cliente`),
- CONSTRAINT `FK_Prestatario_2` FOREIGN KEY (`Numero_prestamo`) REFERENCES `prestamo` (`Numero_prestamo`)
-) ENGINE=InnoDB DEFAULT CHARSET=latin1;

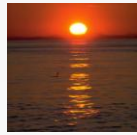


- INSERT INTO `prestatario` (`Nombre_cliente`,`Numero_prestamo`) VALUES
- ('Gómez','p-11'),
- ('López','p-15'),
- ('Fernández','p-16'),
- ('Santos','p-17'),
- ('Valdivieso','p-17'),
- ('Gómez','p-23'),
- ('Pérez','p-93');

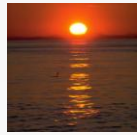


- -- Definition of table `sucursal`
- --

- DROP TABLE IF EXISTS `sucursal`;
- CREATE TABLE `sucursal` (
- `Nombre_sucursal` char(15) NOT NULL,
- `Ciudad_sucursal` char(30) NOT NULL,
- `Activos` decimal(16,2) NOT NULL,
- PRIMARY KEY (`Nombre_sucursal`)
-) ENGINE=InnoDB DEFAULT CHARSET=latin1;

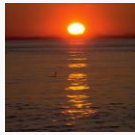


- INSERT INTO `sucursal` (`Nombre_sucursal`,`Ciudad_sucursal`,`Activos`) VALUES
- ('Becerril','Aluche','400.00'),
- ('Centro','Arganzuela','9000.00'),
- ('Collado Mediano','Aluche','8000.00'),
- ('Galapagar','Arganzuela','7100.00'),
- ('Moralzarzal','La Granja','2100.00'),
- ('Navacerrada','Alyche','1700.00'),
- ('Navas Asunción','Alcalá de Henares','3000.00'),
- ('Segovia','Cerceda','3700.00');



Tipos Básicos de Dominios en SQL

- **char(n).** Una cadena de caracteres de longitud fija, con una longitud n especificada por el usuario.
- **varchar(n).** Una cadena de caracteres de longitud variable con una longitud máxima n especificada por el usuario.
- **int.** Un entero (un subconjunto finito de los enteros dependiente de la máquina)
- **smallint.** Un entero pequeño (un subconjunto dependiente de la máquina del tipo de dominio entero).
- **numeric(p,d).** Un número de coma fija, cuya precisión la especifica el usuario. El número está formado por p dígitos, d pertenecen a la parte decimal.
- **real, double precision.** Números de coma flotante y números de coma flotante de doble precisión, con precisión dependiente de la máquina.
- **float(n).** Un número de coma flotante cuya precisión es, al menos, de n dígitos.
- Más tipos de dominios en el Cap 4.



Definición Básica de Esquemas en SQL

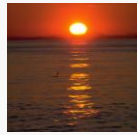
- Las relaciones se definen mediante el comando **create table:**

  create table *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *Donde r* es el nombre de la relación
  - cada A_i es el nombre de un atributo del esquema de la relación *r*
  - D_i es el tipo de dominio de los valores del dominio del atributo A_i

- Ejemplo:

  create table *branch*
  (*branch_name* char(15) **not null,**
  branch_city char(30),
  assets integer)



Restricción de Integridad en la creación de una tabla

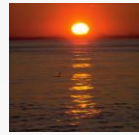
- **not null**
- **primary key** (A_1, ..., A_n)

Ejemplo: Declara *branch_name* como clave primaria de *branch*

.

create table *branch*
(*branch_name* char(15)**,**
branch_city char(30),
assets integer,
primary key (*branch_name*))

primary key declaración de un atributo que garantiza que no tienen valores nulos a partir SQL-92 , era necesaria una declaración explícita en SQL-89



Drop y Alter en la construcción de Tablas

- El comando **drop table** elimina de la BD toda la información de la relación.

  drop table r

- El comando **alter table** se utiliza para añadir atributos a una relación existente:

  alter table *r* **add** *A D*

  donde *A* es el nombre del atributo que se desea añadir y *D* es el dominio del atributo añadido.

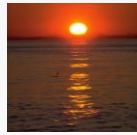
  - Todas las tuplas en la relación son asignadas el valor nulo para el nuevo atributo.

- El comando **alter table** puede también eliminar atributos de una relación:

  alter table *r* **drop** *A*

  donde *A* es el nombre del atributo de la relación *r*

  - Muchos sistemas de BD no permiten el borrado de atributos, aunque si permiten el borrado de tablas completas.



Estructura Básica de las Consultas

- Las BD relacionales están formadas por un conjunto de relaciones.
- La estructura básica de una expresión SQL consta:

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P

  - A_i representa un atributo
  - R_i representa una relación
  - P es un predicado
- Esto es equivalente a al expresion del algebra relacional.

$$\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

- El resultado de una consulta SQL es una relación.

La cláusula select

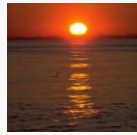
- La cláusula **select** lista los atributos deseados en una consulta resultante
  - Corresponde a la operación proyección del algebra relacional
- Ejemplo: Encuentra los nombres de todas las sucursales (branches) en la relación *loan* :

  select *branch_name*
  from *loan*

  - ***select*** *Nombre_sucursal*
  - ***from*** *banco.prestamo*
  - En el algebra relacional, la consulta sería:

$$\Pi_{branch\_name}(loan)$$

- NOTA: En SQL en los nombres puedes usar mayúsculas o minúsculas
- Ejemplo. *Branch_Name* ≡ *BRANCH_NAME* ≡ *branch_name*
  - Algunas personas usan mayúsculas donde usamos negritas



La cláusula select (Cont.)

- SQL permite duplicados en las relaciones así como, en el resultado de las de las expresiones de consulta.
- Para forzar la eliminación de duplicados, se inserta la palabra **distinct** después de select**.**

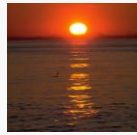
  Encuentra los nombres de todas las sucursales (branches) en la relación *loan* y elimina las tuplas duplicadas

  select distinct *branch_name*
  from *loan*

  - ***select*** *distinct Nombre_sucursal*
  - ***from*** *banco.prestamo*

- La palabra **all** especifica que los duplicados no sean eliminados

  select all *branch_name*
  from bank.*loan*



La cláusula select (Cont.)

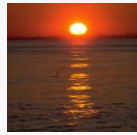
- Un asterisco en la cláusula select denota “todos los atributos”

  select *
  from *loan*

- La cláusula **select** puede contener expresiones aritméticas que involucren la operación , +, –, ∗, y /, y operan en constantes o tuplas
- La consulta:

  select *loan_number, branch_name, amount* ∗ 100
  from *bank.loan*

Da como resultado una relación con el mismo nombre que la relación loan, excepto que el valor del atributo amount es multiplicado por 100.



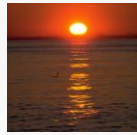
La cláusula where

- La cláusula **where** especifica las condiciones que el resultado debe satisfacer
  - Corresponde a la selección de predicado en el algebra relacional.
- Encontrar todos los números de prestamos (loan number) hechos en la sucursal Perryridge con un importe mayor de $1200.

select *Numero_prestamo*
from *prestamo*
where *Nombre_sucursal* = ‘Centro' **and** *monto* > 1200

- Para comparar resultados se pueden combinar con la conectividad lógica **and, or,** y **not.**

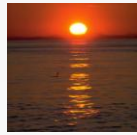
- La comparaciones pueden ser aplicadas a los resultados de expresiones aritméticas.



La cláusula where (Cont.)

- SQL incluye un operador de comparación entre (**between**)
- Ejemplo: encuentra todos números de préstamos de aquellos prestamos cuyo importe este entre \$9,000 y \$10,000 ($\geq$ \$90,000 y $\leq$ \$100,000)

```
SELECT Numero_prestamo
FROM banco.prestamo p
WHERE importe between 900 and 1000;
```



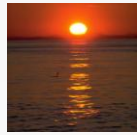
La cláusula from

- La cláusula **from** lista las relaciones involucradas en la consulta
  - Corresponde a la operación del producto cartesiano en el algebra relacional
- Encuentra el producto cartesiano *borrower X loan*

```
select ∗
from borrower, loan
```

- Encuentra el nombre, número de prestamo y el importe del prestamo de la sucursal Centro

```
Select Nombre_cliente, prestatario.Numero_prestamo, importe
        from prestatario, prestamo
        where prestatario.Numero_prestamo  = prestamo.Numero_pretamo  and
                Nombre_sucursal= ‘Centro'
select customer_name, borrower.loan_number, amount
        from borrower, loan
        where   borrower.loan_number = loan.loan_number  and
                branch_name = ‘Centro'
```



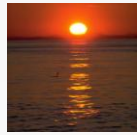
La operación renombrar

- El SQL permite renombrar las relaciones y atributos usando la cláusula **as**:

  old-name **as** *new-name*

- Encuentra nombre, número de prestamo e importe todos los clientes; renombra la columna Numero_prestamo como prestamo_*id.*

select *customer_name, borrower.loan_number* **as** *loan_id, amount*
from *borrower, loan*
where *borrower.loan_number = loan.loan_number*



Variables Tupla

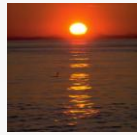
- Variables tupla son definidas en la cláusula **from** mediante la cláusula **as**.
- Encontrar los nombres de los clientes, sus números de prestamos y su importe para todos los clientes que tienen un préstamo en alguna sucursal

select *customer_name, T.loan_number, S.amount*
from *borrower* **as** *T, loan* **as** *S*
where *T.loan_number = S.loan_number*

- Encontrar todos nombres de sucursales que tienen activos mayores que, al menos una sucursal de Aluche.

select distinct *T.branch_name*
from *branch* **as** *T, branch* **as** *S*
where *T.assets* > *S.assets* **and** *S.branch_city* = *'*Aluche*'*

La palabra clave **as** es opcional y puede ser omitida
borrower **as** *T* ≡ *borrower T*



Operaciones con cadena de caracteres

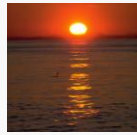
- SQL incluye un operador de coincidencia de caracteres para comparar cadenas. El operador “like” usa patrones que son descritos usando dos caracteres especiales:
  - (%). El % coincide con cualquier subcadena de caracteres
  - (_). El carácter _ coincide con cualquier carácter
- Encontrar el nombre de todos los clientes cuya dirección contenga la subacadena de caracteres “Main” .(Carretas)

```
select customer_name
from customer
where customer_street like '% Main%'
```

- Coincida el nombre “Main%”

```
like 'Main\%' escape '\'
```

- SQL también permite varias funciones sobre cadenas de caracteres tales como
  - concatenación (usando “||”)
  - Conversión de mayúsculas a minúsculas y vice versa
  - Calculo de la longitud de las cadenas, extracción de subacadenas de caracteres.



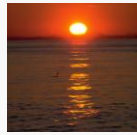
Orden en la presentación de tuplas

- Lista en orden alfabético los nombres de todos los clientes quienes tienen un préstamo en la sucursal Centro

select distinct *customer_name*
from *borrower, loan*
where *borrower loan_number = loan.loan_number* **and**
branch_name = 'Perryridge'
order by *customer_name*

Podemos especificar **desc** para orden descendiente o **asc** para orden ascendente, para cada atributo, el orden ascendente es el de default

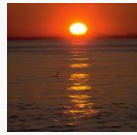
Ejemplo: **order by** *customer_name* **desc**



Duplicados

- En relaciones con duplicados, SQL puede definir cuantas copias de tuplas aparecen en el resultado.
- **Multiconjuntos** versiones de algunas operaciones del algebra relacional- dado un multiconjunto de relaciones r_1 y r_2:

  1. $\sigma_\theta(r_1)$**:** Si hay c_1 copias de la tupla t_1 en r_1, y t_1 satisface la selección σ_θ, entonces hay c_1 copias de t_1 en $\sigma_\theta(r_1)$.
  2. $\Pi_A(r)$**:** Para cada copia de la tupla t_1 en r_1, hay una copia de la tupla $\Pi_A(t_1)$ en $\Pi_A(r_1)$ donde $\Pi_A(t_1)$ denota la proyección de la única tupla t_1.
  3. $r_1 \times r_2$ **:** Si existen c_1 copias de la tupla t_1 en r_1 y c_2 copias de la tupla t_2 en r_2, entonces hay $c_1 * c_2$ copias de la tupla $t_1 . t_2$ en $r_1 \times r_2$



Duplicados (Cont.)

- Ejemplo: Supóngase que las relaciones de multiconjuntos r_1 (*A, B*) y r_2 (*C*) es como sigue:

  $r_1 = \{(1, a)\ (2, a)\}$ $r_2 = \{(2), (3), (3)\}$

- Entonces $\Pi_B(r_1)$ debe ser {(a), (a)}, mientras que $\Pi_B(r_1)$ x r_2 debe ser

  {(*a*,2), (*a*,2), (*a*,3), (*a*,3), (*a*,3), (*a*,3)}

- Semántica de duplicados en SQL:

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

Es equivalente a la versión de multiconjuntos de la expresión:

$$\prod_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

Operaciones sobre conjuntos

- El conjunto de operaciones **union**, **intersect**, y **except** operan sobre relaciones y corresponden a las operaciones del algebra relacional $\cup$, $\cap$, $-$.
- Cada una de las operaciones de anteriores automáticamente elimina duplicados; para retener todos los duplicados se usa la versión de multiconjunto correspondiente **union all**, **intersect all** y **except all**.

  Supóngase que una tupla ocurre *m* veces en *r* y *n veces* en *s, entonces*, esto ocurre:

  - *m* + *n* veces en *r* **union all** *s*
  - min(*m,n)* veces en *r* **intersect all** *s*
  - max(0, *m* – *n)* veces en *r* **except all** *s*

Operaciones sobre conjuntos

- Encuentra todos los clientes que tienen un préstamo, una cuenta o ambas (loan, an account)

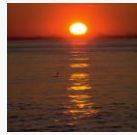
  (select *customer_name* **from** *depositor*)
  union
  (select *customer_name* **from** *borrower)*

- Encuentra todos los clientes que tienen los dos un préstamo, y una cuenta

  (**select** *customer_name* **from** *depositor*)
  intersect
  (select *customer_name* **from** *borrower)*

- Encuentra todos los clientes que tienen préstamo, pero no cuenta.

  (**select** *customer_name* **from** *depositor*)
  except
  (select *customer_name* **from** *borrower)*

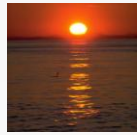


Funciones de Agregación

- Estas funciones operan en multiconjuntos de valores de una columna de una relación y devuelven un solo valor.

  avg: valor de la media
  min: valor mínimo
  max: valor maximo

  sum: suma de valores
  count: número de valores del conjunto



Funciones de Agregación (Cont.)

- Determinar el saldo medio (balance) de las cuentas (account) que se encuentran en la sucursal Perryridge (branch)

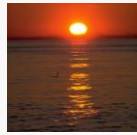
```
select avg (balance)
        from account
        where branch_name = 'Perryridge'
```

- Determine el número de tuplas en la relación cliente (*customer*)

```
select count (*)
        from customer
```

- Encuentre el número de depositantes en el banco (depositors)

```
select count (distinct customer_name)
        from depositor
```

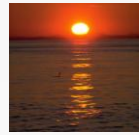


Funciones de Agregación– Group By

- Encuentra el número de depositantes de cada sucursal (branch.)

select *branch_name,* **count (distinct** *customer_name)*
from *depositor, account*
where *depositor.account_number = account.account_number*
group by *branch_name*

Nota: Los atributos en la cláusula **select** afuera de las funciones de agregación debe aparecen en la lista de **group by**

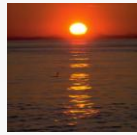


Funciones de Agregación – Having Clause

- Determina los nombres de todas las sucursales donde el saldo promedio (account balance) es mayor de $1,200.

```
select branch_name, avg (balance)
        from account
        group by branch_name
        having avg (balance) > 1200
```

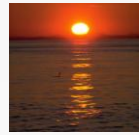
Nota: Los predicados en la cláusula **having** son usados después de la formación de grupos mientras que en la cláusula **where** son usados antes de la formación de grupos.



Valores Nulos

- Es posible para las tuplas tener un valor nulo, denotado por *nulo*, para algunos de sus atributos.
- *Nulo* significa un valor desconocido o no existente.
- Cuando predicado es **null puede ser usado para checar valores nulos**.
  - Ejemplo: Encuentra todos los números de préstamo que aparecen en la relación préstamo que tienen un importe nulo. (null values for *amount)*

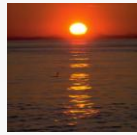
    select *loan_number*
    from *loan*
    where *amount* **is null**
- El resultado de cualquier operación aritmética que involucre al *nulo* es *nulo.*
  - Ejemplo: 5 + *null* regresa null
- Las funciones de agregación simplemente ignoran los valores nulos
  - Mas en la siguiente dispositiva



Valores nulos y los 3 valores lógicos

- Comparaciones con valores devuelven el valor especial de verdad: desconocido.
  - Ejemplo*: 5 < null o null <> null o null = null*
- Se definen las 3 operaciones lógicas tratan el valor lógico *desconocido*:
  - O: (desconocido **o** cierto) = *cierto*,
    (desconocido **o** *falso*) = desconocido
    (*desconocido* **o** *desconocido*) = *desconocido*
  - Y: (*cierto* **y** *desconocido*) = *desconocido,*
    (*falso* **y** *desconocido*) = *falso,*
    (*desconocido* **y** *desconocido*) = *desconocido*
  - NO*:* (**no** *desconocido*) = *desconocido*
  - "*P* **es** *desconocido*" se evalúa cierto si el predicado devuelve el valor *desconocido*

El resultado (cláusula **where)** de un predicado se trata como falso si se evalúa como desconocido

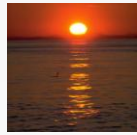


Valores nulos y agregados

- Total de todos los importes (amount) de préstamo (loan)

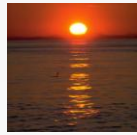
  select sum (*amount*)
  from *loan*

  - El enunciado de arriba ignora los importes nulos
  - El resultado es nulo si hay no no-nulo importe
- Todas las operaciones de agregación excepto **count(*)** ignoran las tuplas con valores nulos en los atributos de agregación.



Subconsultas anidadas

- SQL provee un mecanismo para las subconsultas anidadas.
- Una subconsulta (**subquery)** es una expresión **select-from-where** que es anidada con otra consulta.
- Un uso común de las subconsultas es para pruebas de conjuntos de membresía, conjuto de comparaciones y conjunto de cardinalidad.



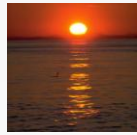
Ejemplo de Consultas

- Encuentra todos los clientes (customers) quienes tienen ambas cuenta y préstamo en el banco (account and loan)

```
select distinct customer_name
        from borrower
        where customer_name in (select customer_name
                                from depositor )
```

- Encuentra todos los clientes (customers) quienes tienen un préstamo y no una cuenta en el banco.

```
select distinct customer_name
        from borrower
        where customer_name not in (select customer_name
                                    from depositor )
```

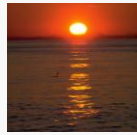


Ejemplo de Consultas

- Encuentra todos los clientes (customers) quienes tienen ambas cuenta y préstamo en el banco (account and loan) en la sucursal Perryridge (branch)

```
select distinct customer_name
        from borrower, loan
        where borrower.loan_number = loan.loan_number and
          branch_name = 'Perryridge'  and
          (branch_name, customer_name ) in
                (select branch_name, customer_name
                  from depositor, account
                  where depositor.account_number =
                      account.account_number )
```

- Nota: La consulta de arriba se puede escribir de una manera mas simple. La formulación de arriba es simplemente para ilustrar las características del SQL.



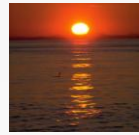
Comparación de Conjuntos

- Determinar el nombre de todas las sucursales que poseen activos (*assets*) mayores que, al menos, una sucursal de Brooklyn.

```
select distinct  T.branch_name
        from branch as T, branch as S
        where  T.assets > S.assets and
                S.branch_city = 'Brooklyn'
```

- La misma consulta usando > **some** (cláusula)

```
select branch_name
        from branch
        where assets > some
                (select assets
                 from branch
                 where branch_city = 'Brooklyn')
```



Definición de la Cláusula Some

- F <comp> **some** $r \Leftrightarrow \exists\, t \in r$ tal que (F <comp> t)
  donde <comp> puede ser: $<,\ \leq,\ >,\ =,\ \neq$

(5 < **some** | 0 / 5 / 6 |) = true (se lee: 5 < alguna tupla en la relación)

(5 < **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (since 0 ≠ 5)

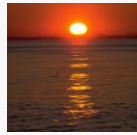
(= **some**) ≡ **in**

Sin embargo, (≠ **some**) ≢ **not in**

Example Query

- Encuentra los nombres de todas las sucursales que tienen los activos mas grandes en importe que todas las sucursales localizadas en Brooklyn.
- Find the names of all branches that have greater assets than all branches located in Brooklyn.

```
select branch_name
        from branch
        where assets > all
                (select assets
                from branch
                where branch_city = 'Brooklyn')
```



Definición de la Cláusula all

- F <comp> **all** $r \Leftrightarrow \forall\ t \in r$ (F <comp> *t)*

(5 < **all** | 0, 5, 6 |) = false

(5 < **all** | 6, 10 |) = true

(5 = **all** | 4, 5 |) = false

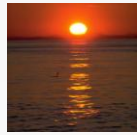
(5 ≠ **all** | 4, 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) ≡ **not in**

However, (= **all**) ≢ **in**

Comprobación de relaciones vacías

- El constructor exists devuelve el valor true si el argumento de la subconsulta no es vacía.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

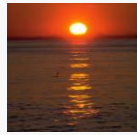


Ejemplo de consulta

- Encuentre todos los clientes quienes tienen una cuenta en todas las sucursales localizadas en Brooklyn.

```
select distinct S.customer_name
        from depositor as S
        where not exists (
                (select branch_name
                from branch
                where branch_city = 'Brooklyn')
                except
                (select R.branch_name
                from depositor as T, account as R
                where T.account_number = R.account_number and
                        S.customer_name = T.customer_name ))
```

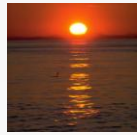
- Nota que $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
- *Nota: No se puede escribir esta consulta usando* = **all y sus variantes**



Comprobación de la ausencia de tuplas duplicadas

- La construcción **unique** comprueba si las subconsultas tienen tuplas duplicadas en su resultado.
- Determinar todos los clientes quienes tienen, a lo sumo, una cuenta en la sucursal Perryridge (branch).

```
select T.customer_name
from depositor as T
where unique (
        select R.customer_name
        from account, depositor as R
        where T.customer_name = R.customer_name and
              R.account_number = account.account_number and
              account.branch_name = 'Perryridge')
```

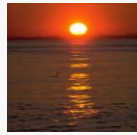


Ejemplo de consulta

- Determina todos los clientes quienes tienen al menos dos cuentas en la sucursal Perryridge (branch).

```
select distinct T.customer_name
from depositor as T
where not unique (
    select R.customer_name
    from account, depositor as R
    where T.customer_name = R.customer_name and
          R.account_number = account.account_number and
          account.branch_name = 'Perryridge')
```

- La variable de un nivel exterior es conocida como **variable de correlación.**

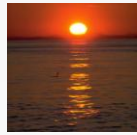


Relaciones Derivadas

- SQL permite el uso de expresiones de subconsulta en las cláusulas From.
- Encuentra la media del balance de cuenta de todas las sucursales donde la media del balance sea superior a $1200.

```
select branch_name, avg_balance
from (select branch_name, avg (balance)
        from account
        group by branch_name )
      as branch_avg ( branch_name, avg_balance )
where avg_balance > 1200
```

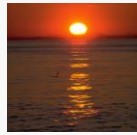
Nota que no necesitamos usar la cláusula having, ya que la subconsulta calcula el saldo medio y su resultado se denomina branch avg en la cláusula; los atributos de la branch avg pueden ser usados directamente en la clásusula where



Cláusula With

- La cláusula **with** proporciona una forma de definir vistas temporales cuya definición solo esta disponible para la consulta en la que aparece la cláusula.
- Encuentra todas las cuentas con el saldo máximo.

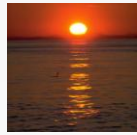
with *max_balance* (*value*) **as**
    select max (*balance*)
    from *account*
select *account_number*
from *account, max_balance*
where *account.balance = max_balance.value*



Consultas complejas usando la cláusula With

- Determinar todas las sucursales donde el depósito total de las cuentas es mayor que la media de los depósitos totales de la cuentas de todas las sucursales.
- Find all branches where the total account deposit is greater than the average of the total account deposits at all branches.

```
with branch_total (branch_name, value) as
        select branch_name, sum (balance)
        from account
        group by branch_name
with branch_total_avg (value) as
        select avg (value)
        from branch_total
select branch_name
from branch_total, branch_total_avg
where branch_total.value >= branch_total_avg.value
```

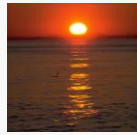


Vistas

- En algunos casos, no es deseable que todos los usuarios vean el modelo lógico completo (esto es, las relaciones reales almacenadas en la BD)
- Considere una persona que necesita saber, el nombre del cliente, número de préstamo y nombre de la sucursal pero no el importe del préstamo. Esta persona debería ver la relación descrita en SQL como:

(**select** *customer_name, borrower.loan_number, branch_name*
from *borrower, loan*
where *borrower.loan_number = loan.loan_number*)

- Una **view** proporciona un mecanismo para ocultar cierto tipo de información.
- Las relaciones que no forman parte del modelo lógico pero se hacen visibles a los usuarios como relaciones virtuales se denominan **view**.

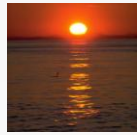


Definición de Vistas

- Una vista es definida usando la instrucción **create view la cual tiene la siguiente forma:**

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

  donde < expresión de consulta > es cualquier expresión legal de consulta en SQL. El nombre de la vista se representa por *v.*
- Una vez definida la vista, el nombre de la vista puede ser usado para referirse a la relación virtual que genera.
- Cuando una vista es creada, la consulta de expresión es almacenada en la BD; la expresión es sustituida en las consultas usando vistas.



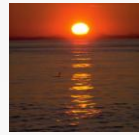
Ejemplo de consultas

- Una vista que consista de sucursales (branches) y sus clientes

```
create view all_customer as
        (select branch_name, customer_name
         from depositor, account
        where depositor.account_number =
                account.account_number )
        union
        (select branch_name, customer_name
        from borrower, loan
        where borrower.loan_number = loan.loan_number )
```

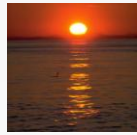
- Encuentra todos los clientes de la sucursal Perryridge (branch)

```
select customer_name
        from all_customer
        where branch_name = 'Perryridge'
```



Vistas definidas en función de otras

- Una vista puede ser utilizada en las expresiones que definen otras vistas.
- Una relación vista v_1 se dice ser *directamente dependiente* de una relación vista v_2 si v_2 es utilizada en la expresión que define v_1
- Una relación vista v se dice ser *recursiva* si esta depende de si misma.

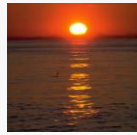


Expansión de Vistas

- Es una manera de definir el significado de las vistas definidas en términos de otras.
- Sea una vista v_1 definida mediante una expresión e_1 que puede contener a su vez relaciones de vistas.
- La expansión de vista de una expresión repite la etapa de sustitución de la manera siguiente:

  repeat
  Encuentra todas las relaciones de vistas v_i de e_1
  Sustituir la relación de vistas v_i por la expresión que define v_i
  until no queden mas relaciones de vistas en e_1

- Mientras las definiciones de las vistas no sean recursivas, el bucle concluirá.



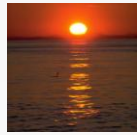
Modificación BD– Deletion

- Borrar todas las tuplas de la sucursal Perryridge (branch)

  delete from *account*
  where *branch_name* = 'Perryridge'

- Borrar todas las cuentas de cada sucursal localizada en la ciudad ‘Needham’.

  delete from *account*
  where *branch_name* **in** (**select** *branch_name*
  from *branch*
  where *branch_city* = 'Needham')



Ejemplo de Consulta

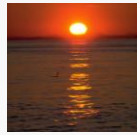
- Borrar los registros de todas las cuentas con balances menores al promedio del banco.

delete from *account*
where *balance* < (**select avg** (*balance*)
from *account*)

- Problema: si borramos las tuplas de deposito (deposit), el promedio del balance cambia.
- Solución usando SQL:

  1. Primero calcule el **avg** balance y luego encuentre las tuplas a borrar.

  2. Siguiente, borre todas las tuplas encontradas arriba (sin recalcular el **avg** o recomprobar las tuplas)



Modificación de la BD – Insertion

- Añade una nueva tupla a cuenta (*account)*

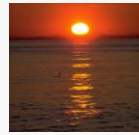
  insert into *account*
  values ('A-9732', 'Perryridge', 1200)

O equivalentemente

insert into *account* (*branch_name, balance, account_number*)
values ('Perryridge', 1200, 'A-9732')

- Añade una tupla a cuenta (account) con balance desconocido (null)

  insert into *account*
  values ('A-777','Perryridge', *null*)



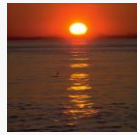
Modificación de la BD– Insertion

- Supóngase que se desea ofrecer una nueva cuenta de ahorro con $200 como regalo a todos los clientes con préstamos concedidos en la sucursal Perryridge. Se usará el número de préstamo como número de esta cuenta de ahorro.

```
insert into account
    select loan_number, branch_name,  200
    from loan
    where branch_name = 'Perryridge'
insert into depositor
    select customer_name, loan_number
    from loan, borrower
    where branch_name = 'Perryridge'
            and loan.account_number = borrower.account_number
```

- La declaración de **select from where** es evaluada totalmente antes de que sus resultados sean insertados dentro de la relación (de otra manera lo siguiente puede causar problemas

  insert into *table*1 **select** * **from** *table*1)



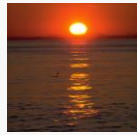
Modificación de la BD– Updates

- Las cuentas con saldos superiores a $10,000 reciben 6% de interés, mientras que el resto recibe un 5%
  - Escribe dos actualizaciones:

update *account*
set *balance* = *balance* ∗ 1.06
where *balance* > 10000

update *account*
set *balance* = *balance* ∗ 1.05
where *balance* ≤ 10000

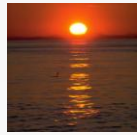
  - El orden es importante
  - Se puede hacer mejor usando la declaración **case** (siguiente slide)



Declaración Case para actualizaciones condicionales

- La misma consulta anterior: Las cuentas con saldos superiores a $10,000 reciben 6% de interés, mientras que el resto recibe un 5%.

```
update account
set balance =  case
                  when balance <= 10000 then balance *1.05
                  else   balance * 1.06
               end
```



Actualización de vistas

- Crea una vista de todos los datos de préstamo de la relación préstamo, ocultando la cantidad (amount)

  create view *loan_branch* **as**
  select *loan_number, branch_name*
  from *loan*

- Añade una nueva tupla para l*oan_branch*

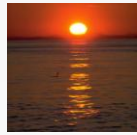
  insert into *loan_branch*

  values ('L-37‘, 'Perryridge‘)

Esta inserción debe ser representada por la inserción de la tupla

('L-37', 'Perryridge', *null*)

En la relación préstamo (*loan)*



Actualizaciones a través de vistas (Cont.)

- Algunas de las actualizaciones a través de vistas son imposible de traducir en actualizaciones de las relaciones en la BD.
  - **create view** *v* **as**
    select *loan_number, branch_name, amount*
    from *loan*
    where *branch_name* = ‘Perryridge’

    insert into *v* **values ('**L-99*',*'Downtown*', '23'***)**
- Otras no pueden ser traducidos como en

  insert into *all_customer*

  values *('*Perryridge', *'*John')
  - Se tiene que escoger préstamo o cuenta, y crear una nuevo número préstamo/cuenta!
- La mayoría de SQL implementaciones permiten actualizaciones de vistas simples (sin agregados) definidos en una sola relación de la BD.

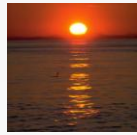


Figure 3.3: Tuplas insertadas en *loan* y *borrower*

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500
null	*null*	1900

loan

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17
Johnson	*null*

borrower

Reunión de relaciones**

- **Operaciones de reunión** toma dos relaciones y devuelve como resultado otra relación.
- Estas operaciones de reunión externa son típicamente usadas en la subconsulta de expresiones en la cláusula **from**, aunque se pueden utilizar en cualquier lugar en el que se pueda utilizar una relación.
- **Condición de reunión** – define cuales tuplas en dos relaciones a casar, y que atributos que se incluyen en el resultado de reunión.
- **Tipo de reunión** – define la manera de tratar las tuplas de cada relación que no casan con ninguna tupla de la otra relación (de acuerdo con la condición de reunión)

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Reunión de Relaciones –Ejemplos

- Relación *loan*
- Relación *borrower*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

- Nota: que la información de prestatario (borrower) no se encuetra para L-260 y la información de préstamo (loan) L-155 no se encuntra.

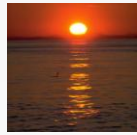
Reunión de Relaciones –Ejemplos

- *loan* **inner join** *borrower* **on**
  loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

- *loan* **left outer join** *borrower* **on**
  loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*



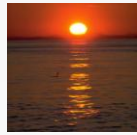
Reunión de Relaciones –Ejemplos

- *loan* **natural inner join** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

- *loan* **natural right outer join** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes



Reunión de Relaciones –Ejemplos

- *loan* **full outer join** *borrower* **using** (*loan_number*)

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

- Find all customers who have either an account or a loan (but not both) at the bank.

select *customer_name*
from (*depositor* **natural full outer join** *borrower*)
where *account_number* **is null or** *loan_number* **is null**

Tarea;
Ejercicios prácticos
3.1-3.7 5ta edición

Fin del Capítulo 3

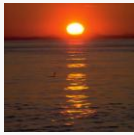


Figure 3.4: The *loan* and *borrower* relations

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower