

## SQL – Structured Query Language

Elementos de Bases de Datos  
Bases de Datos  
Universidad Nacional del Sur - 2014

1

## Lenguaje SQL

- SQL: Structured Query Language
- SQL es un lenguaje de manipulación de datos (**DML**) y un lenguaje de definición de datos (**DDL**).
- Es un lenguaje **Procedural** muy cercano al algebra relacional.
- Existen muchos dialectos y varios estándares SQL89, SQL92, SQL99, SQL2003.

2

## Capacidades del Lenguaje

- **DDL** - Lenguaje de definición de datos.
  - Definición de relaciones y vistas.
  - Instrucciones para autorización.
  - Definición de reglas de integridad.
- **DML** - Lenguaje de manipulación de datos.
  - Consultar datos almacenados.
  - Modificar el contenido de los datos almacenados.

3

## Lenguaje de Manipulación de Datos (DML)

4

## Recuperando el contenido de la Base de Datos (consultas)

5

## Estructura Básica de una consulta SQL

- La estructura básica consiste de tres cláusulas: **SELECT**, **FROM** y **WHERE**.
- SELECT**: atributos deseados.  
*proyección* del álgebra relacional.
- FROM**: una o más tablas.  
*producto cartesiano* del álgebra relacional.
- WHERE**: condición sobre las tóplas.  
*selección* del álgebra relacional.

6

## Esquema utilizado en ejemplos

- PELICULA(titulo, anio, duracion, encolor, estudio, director)
  - Estudio llave foránea de Estudio(ID)
  - Director llave foránea de Director(cert)
- DIRECTOR(nombre, direccion, cert)
- ACTUA(titulo, anio, actornombre)
  - Titulo y Anio llave foránea de Pelicula(titulo, anio)
  - Actornombre llave foránea de Actor
- ACTOR(nombre, direccion, sexo, fechaNac)
- ESTUDIO(nombre, direccion, id)

7

## Proyección

ACTUA		
titulo	Anio	actorNombre
TITANIC	1997	Leonardo Di Caprio
EL SEÑOR DE LOS ANILLOS: EL RETORNO DEL REY	2003	Elijah Wood
Piratas del Caribe: El cofre del hombre muerto	2006	Johnny Depp
Harry Potter y la piedra filosofal	2001	Daniel Radcliffe
El Señor de los anillos: Las dos torres	2002	Elijah Wood
La guerra de las galaxias. Episodio I: La amenaza fantasma	1999	Ewan McGregor

SELECT titulo, Anio  
FROM ACTUA;

ACTUA	
titulo	Anio
TITANIC	1997
EL SEÑOR DE LOS ANILLOS: EL RETORNO DEL REY	2003
Piratas del Caribe: El cofre del hombre muerto	2006
Harry Potter y la piedra filosofal	2001
El Señor de los anillos: Las dos torres	2002
La guerra de las galaxias. Episodio I: La amenaza fantasma	1999

8

## El \* en la cláusula SELECT

- Cuando en la cláusula FROM existe sólo una relación, un \* en la cláusula SELECT significa "Todos los atributos de esta relación"

```
SELECT titulo, Anio, actorNombre
FROM ACTUA;
```

Es equivalente a

```
SELECT *
FROM ACTUA;
```

ACTUA		
titulo	Anio	actorNombre
TITANIC	1997	Leonardo Di Caprio
EL SEÑOR DE LOS ANILLOS: EL RETORNO DEL REY	2003	Elijah Wood
Piratas del Caribe: El cofre del hombre muerto	2006	Johnny Depp
Harry Potter y la piedra filosofal	2001	Daniel Radcliffe
El Señor de los anillos: Las dos torres	2002	Elijah Wood
La guerra de las galaxias. Episodio I: La amenaza fantasma	1999	Ewan McGregor

## Elementos repetidos

```
SELECT actorNombre
FROM ACTUA;
```

ACTUA		
titulo	Anio	actorNombre
TITANIC	1997	Leonardo Di Caprio
EL SEÑOR DE LOS ANILLOS: EL RETORNO DEL REY	2003	Elijah Wood
Piratas del Caribe: El cofre del hombre muerto	2006	Johnny Depp
Harry Potter y la piedra filosofal	2001	Daniel Radcliffe
El Señor de los anillos: Las dos torres	2002	Elijah Wood
La guerra de las galaxias. Episodio I: La amenaza fantasma	1999	Ewan McGregor

ACTUA
actorNombre
Leonardo Di Caprio
Elijah Wood
Johnny Depp
Daniel Radcliffe
Elijah Wood
Ewan McGregor

10

## Semántica de Bolsa (Bag)

- Las cláusulas SELECT, FROM, WHERE utilizan **semántica Bag**, es decir, que no se eliminan las tûplas duplicadas.
- Para utilizar una **semántica de conjunto (Set)** es necesario utilizar la cláusula DISTINCT luego del SELECT.

```
SELECT DISTINCT actorNombre
FROM Actua;
```

11

## Selección

```
SELECT DISTINCT actorNombre
FROM ACTUA
WHERE Anio > 2000;
```

ACTUA		
titulo	Anio	actorNombre
TITANIC	1997	Leonardo Di Caprio
EL SEÑOR DE LOS ANILLOS: EL RETORNO DEL REY	2003	Elijah Wood
Piratas del Caribe: El cofre del hombre muerto	2006	Johnny Depp
Harry Potter y la piedra filosofal	2001	Daniel Radcliffe
El Señor de los anillos: Las dos torres	2002	Elijah Wood
La guerra de las galaxias. Episodio I: La amenaza fantasma	1999	Ewan McGregor

ACTUA
actorNombre
Elijah Wood
Johnny Depp
Daniel Radcliffe

12

## Selección de t pulas en SQL

- Se implementa a trav s de las expresiones condicionales en el **WHERE**.
- Las expresiones se arman con los **operadores de comparaci n** `>`, `<`, `=`, `<>`, `>=`, `<=` y los **operadores l gicos** **AND**, **OR** y **NOT**.
- Los valores que se comparan pueden incluir atributos de las relaciones mencionadas en el FROM y/o constantes.
- Pueden usarse operadores aritm ticos (+, -, etc) siempre que se respeten los tipos de los datos a comparar.

13

## Ejemplos: Selecci n

PELICULA(titulo, anio, duracion, encolor, estudio, director)

```
SELECT titulo
FROM Pelicula
WHERE anio > 1970 AND NOT encolor;
```

```
SELECT titulo, encolor, duracion
FROM Pelicula
WHERE (anio > 1970 OR duracion < 90)
AND estudio = 2;
```

14

## Comparaci n de Strings

- Pueden realizarse comparaciones con operadores relacionales `<`, `>`, `>=`, `<=`, `=`, `<>` seg n el orden lexicogr fico.
- Tambi n comparaciones de patrones:
  - `<Atributo> LIKE <patr n>`
  - `<Atributo> NOT LIKE <patr n>`
- El patr n es un string que puede contener dos caracteres especiales `%` y `_`

15

## Comparaci n con LIKE

- `s LIKE p [ESCAPE '<caracter>']`
- Los caracteres distintos de `%` y `_` deben corresponderse con ellos mismos.
- El caracter `%` (en `s`) puede corresponderse con cualquier cadena de 0 o m s caracteres (en `p`).
- El caracter `_` (en `s`) se corresponde con un  nico caracter cualquiera (en `p`)

16

## Consulta Simple: LIKE

ACTUA		
titulo	Anio	actorNombre
TITANIC	1997	Leonardo Di Caprio
EL SEÑOR DE LOS ANILLOS: EL RETORNO DEL REY	2003	Elijah Wood
Piratas del Caribe: El cofre del hombre muerto	2006	Johnny Depp
Harry Potter y la piedra filosofal	2001	Daniel Radcliffe
El Señor de los anillos: Las dos torres	2002	Elijah Wood
La guerra de las galaxias. Episodio I: La amenaza fantasma	1999	Ewan McGregor

```
SELECT titulo, Anio
FROM ACTUA
WHERE titulo
LIKE '%anillos%';
```

ACTUA	
titulo	Anio
El Señor de los anillos: Las dos torres	2002

SQL es *case-insensitive* excepto en los strings. MySQL se comporta según el conjunto de caracteres y regionalización (utilizar sino el operador LIKE BINARY).

17

## Mejorando la Salida (ORDER BY)

- Para mostrar las túplas resultantes ordenadas de acuerdo a ciertos atributos se puede utilizar la cláusula:  
**ORDER BY** < lista de atributos separados por comas >
- Si se desea que el ordenamiento sobre un cierto atributo sea de forma descendente, se agregar la clausula **DESC** a continuación de dicho atributo.
- Por defecto es ascendente y con prioridad izquierda a derecha.

18

## Mejorando la Salida (ORDER BY)

- Ejemplo de ordenamiento:

```
SELECT titulo, actornombre
FROM ACTUA
WHERE anio > 2000
ORDER BY actornombre, titulo DESC;
```

- Podemos mejorar la legibilidad de los atributos al mostrar los resultados podemos renombrar los atributos mediante la clausula "**AS** <nombre>".

19

## Mejorando la Salida (AS)

```
SELECT titulo AS 'Título', Anio AS 'Año', actorNombre AS Actor
FROM ACTUA
WHERE anio > 2000;
```

ACTUA		
Título	Año	Actor
EL SEÑOR DE LOS ANILLOS: EL RETORNO DEL REY	2003	Elijah Wood
Piratas del Caribe: El cofre del hombre muerto	2006	Johnny Depp
Harry Potter y la piedra filosofal	2001	Daniel Radcliffe
El Señor de los anillos: Las dos torres	2002	Elijah Wood

20

## Expresiones en SELECT

PELICULA(titulo, anio, duracion, encolor, estudio, director#)

### ■ Expresiones

```
SELECT titulo, hour(duracion)*60+minute(duracion) AS
'Duración min.' FROM Pelicula;
```

### ■ Constantes

```
SELECT titulo, 'B&W' AS color
FROM Pelicula
WHERE NOT encolor;
```

21

## Múltiples Relaciones

- Si después de la palabra reservada **FROM** se enumera más de una relación, SQL hace el **producto cartesiano** entre ellas.
- Un producto cartesiano sin clausula WHERE generalmente da resultados sin significado semántico.

```
SELECT titulo, nombre AS director
FROM Pelicula, Director;
```

22

## Consultas sobre múltiples relaciones

- Supongamos querer conocer el nombre del director de la saga "El señor de los Anillos".

PELICULA(titulo, anio, duracion, encolor, estudio, director)  
DIRECTOR(nombre, direccion, cert)

```
SELECT DISTINCT nombre as 'Nombre Director'
FROM PELICULA, DIRECTOR
WHERE titulo LIKE '%anillos%' AND
director = cert;
```

23

## Consultas sobre múltiples relaciones

- Supongamos querer conocer el nombre del director y actores que vivan juntos.

ACTOR(nombre, direccion, sexo, fechaNac)  
DIRECTOR(nombre, direccion, cert)

```
SELECT DISTINCT nombre, nombre
FROM ACTOR, DIRECTOR
WHERE direccion = direccion;
```

- Para distinguir atributos con el mismo nombre utilizamos "**<relación>.<atributo>**"

```
SELECT DISTINCT ACTOR.nombre, DIRECTOR.nombre
FROM ACTOR, DIRECTOR
WHERE ACTOR.direccion = DIRECTOR.direccion;
```

24

## Consultas sobre múltiples relaciones

- Supongamos querer conocer el nombre de todos los pares de actores que vivan juntos.  
ACTOR(nombre, direccion, sexo, fechaNac)
- SQL permite establecer un alias para las relaciones de la clausula FROM.

```
SELECT Star1.nombre, Star2.nombre
FROM ACTOR Star1, ACTOR Star2
WHERE Star1.direccion = Star2.direccion AND
      Star1.nombre < Star2.nombre;
```

¿Qué hubiese sucedido si en lugar de `Star1.nombre < Star2.nombre` hubiesemos puesto `Star1.nombre <> Star2.nombre`? ¿Se podría solucionar con la clausula `DISTINCT`?

25

## Semántica Operacional

- Una consulta típica SQL es de la forma:

```
select A1, A2, ..., An
from r1, r2, ..., rm
[ where P ]
```

- $A_i$  representa un atributo
- $r_i$  representa una relación
- $P$  es un predicado.
- Esta consulta se **expresa** y **resuelve** como en Algebra Relacional:

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

26

## Unión, Intersección, Diferencia

- Las operaciones de unión, intersección y diferencia sobre relaciones se expresan de la siguiente forma:

( subconsulta ) **UNION** ( subconsulta )  
 ( subconsulta ) **INTERSECT** ( subconsulta )  
 ( subconsulta ) **EXCEPT** ( subconsulta )

- Las cláusulas **UNION**, **INTERSECT** y **EXCEPT** permiten vincular dos subconsultas **con la misma estructura** (esquema).

27

## Unión, Intersección, Diferencia

ACTOR(nombre, direccion, sexo, fechaNac)  
 DIRECTOR(nombre, direccion, cert#)

ACTOR			
nombre	direccion	sexo	fechaNac
Leonardo Di Caprio	1st Av. 100	M	Null
Elijah Wood	2nd Av. 200	M	Null
Johnny Depp	3rd Av. 300	M	Null
Daniel Radcliffe	1st Av. 100	M	Null
Ewan McGregor	5th Av. 500	M	null

DIRECTOR		
Nombre	direccion	Cert#
Ewan McGregor	5th Av. 500	101
Elijah Wood	3rd Av. 300	102
Ewan McGregor	5th Av. 500	103

```
(SELECT nombre, direccion
FROM ACTOR
WHERE sexo = 'M')
```

INTERSECT

```
(SELECT nombre, direccion
FROM director);
```

Intersect	
Nombre	direccion
Ewan McGregor	5th Av. 500

28

## Semántica de Unión, Intersección y Diferencia

- A diferencia de la sentencia SELECT FROM WHERE que utiliza la semántica de bolso (Bag), las operaciones de unión, intersección y diferencia utilizan la **semántica de conjunto** (Set). Es decir que todas las **túplas duplicadas son eliminadas**.
- Para evitar eliminar duplicados es necesario escribir la palabra reservada **ALL** luego de **UNION, INTERSECT** o **EXCEPT**
- **MySQL** solo implementa UNION.

29

## Subconsultas

Ver. 13.2.10

- Una consulta que es parte de otra consulta es llamada **subconsulta**.
- Las subconsultas se escriben entre paréntesis
- Las subconsultas pueden utilizarse en:
  - UNION, INTERSECT, EXCEPT
  - Cláusula WHERE
    - Retornando un valor escalar.
    - Retornando una relación.
  - Cláusula FROM
  - En algunos dialectos en la cláusula SELECT

30

## Subconsultas como constantes

- Si una subconsulta garantiza producir **una única túpla**, entonces la subconsulta puede ser utilizada **como un valor**.
  - Usualmente, estamos interesados en un único atributo, que además utilizando las llaves y otra información podemos deducir que se producirá un único valor.
  - Si la subconsulta produce más de una túpla o ninguna, se produce un error en tiempo de ejecución.

31

## Subconsultas como constantes

- Supongamos querer conocer el nombre del director de la saga "El señor de los Anillos".

PELICULA(titulo, anio, duracion, encolor, estudio, director)  
DIRECTOR(nombre, direccion, cert)

```
SELECT DISTINCT nombre
FROM PELICULA, DIRECTOR
WHERE titulo LIKE '%anillos%' AND director = cert;
```

```
SELECT nombre
FROM DIRECTOR
WHERE cert = (
    SELECT DISTINCT director
    FROM PELICULA
    WHERE titulo LIKE '%anillos%'
);
```

32

## Subconsultas que producen relaciones

- Cuando se utiliza una subconsulta en la cláusula WHERE y la subconsulta retorna una relación, SQL provee una serie de operadores para obtener un resultado booleano.
- Sea  $s$  un valor escalar (*valor atómico que puede aparecer como componente de una tupla*) y  $R$  una relación **unária**.
- **EXISTS**  $R$  es verdadero si  $R$  **no es vacío**.
- **NOT EXISTS**  $R$  es verdadero si  $R$  **es vacío**.

33

## Subconsultas que producen relaciones

- $s$  **IN**  $R$  es verdadero si el atributo  $s$  **está en**  $R$ . También puede ser precedido por NOT ( $s$  **NOT IN**  $R$ ).
- $s$  **> ALL**  $R$  es verdadero si el atributo  $s$  es mayor que **todos** los elementos de  $R$ .
- Pueden utilizarse cualquiera de los otros operadores de comparación en lugar de  $>$ .
- $s$  **> ANY**  $R$  es verdadero si el atributo  $s$  es mayor que **alguno** de los elementos de  $R$ . También se puede utilizar **SOME**.

MySQL reporta que las subconsultas con ANY son preferibles en eficiencia a las ALL.

En algunos dialectos existe el operador CONTAINS equiv. a la división en AR. 34

## Subconsultas que producen relaciones

- Si  $s$  es una tupla y  $R$  una relación con la misma aridad de  $s$  entonces los operadores antes mencionados también podrán ser utilizados.

```
SELECT nombre
FROM DIRECTOR
WHERE cert IN (SELECT DISTINCT director
              FROM PELICULA
              WHERE (titulo, anio) IN
                  ( SELECT titulo, anio
                    FROM ACTUA
                    WHERE actorNombre = 'Johnny Depp' )
              );
```

35

## Subconsulta en cláusula FROM

- En lugar de realizar consultas temporales, se puede utilizar subconsultas dentro de la cláusula FROM.
- Usualmente se le debe dar un alias a dicha subconsulta.

```
SELECT nombre
FROM DIRECTOR D, ( SELECT DISTINCT director
                  FROM PELICULA P, ACTUA A
                  WHERE P.titulo = A.titulo AND
                        P.anio = A.anio AND
                        actorNombre = 'Johnny Depp'
                  ) DIR
WHERE D.cert = DIR.director;
```

36

## Subconsultas

- Las mismos resultados pueden se pueden obtener con distintas consultas.
- Las consultas anidadas son más eficientes en tiempo que las consultas que tienen muchas relaciones siguiendo a la cláusula FROM.

37

## Expresiones JOIN

- A partir de SQL99 se provee diversas formas de JOIN similares al join del algebra relacional.
  - Pero utilizan semántica bag y no semántica set.
  - Cada dialecto de SQL provee diferentes palabras reservadas.

38

## Theta Join

- `R JOIN S ON <condición>`  
es un theta-join con <condición> para realizar la selección.

```
SELECT DISTINCT director
FROM PELICULA P JOIN ACTUA A ON P.titulo = A.titulo
AND P.anio = A.anio
WHERE actorNombre = 'Johnny Depp';
```

39

## JOIN Natural

- Este join difiere del Theta-join en:
  1. La condición del join es implícita, igualando todos los atributos de ambas relaciones que tengan el mismo nombre.
  2. Sólo uno de los atributos igualados es dejado como resultado.

```
DIRECTOR(nombre, direccion, cert)
ACTOR(nombre, direccion, sexo, fechaNac)
```

```
SELECT * FROM ACTOR NATURAL JOIN DIRECTOR;
```

OUTPUT(nombre, direccion, sexo, fechaNac, cert)

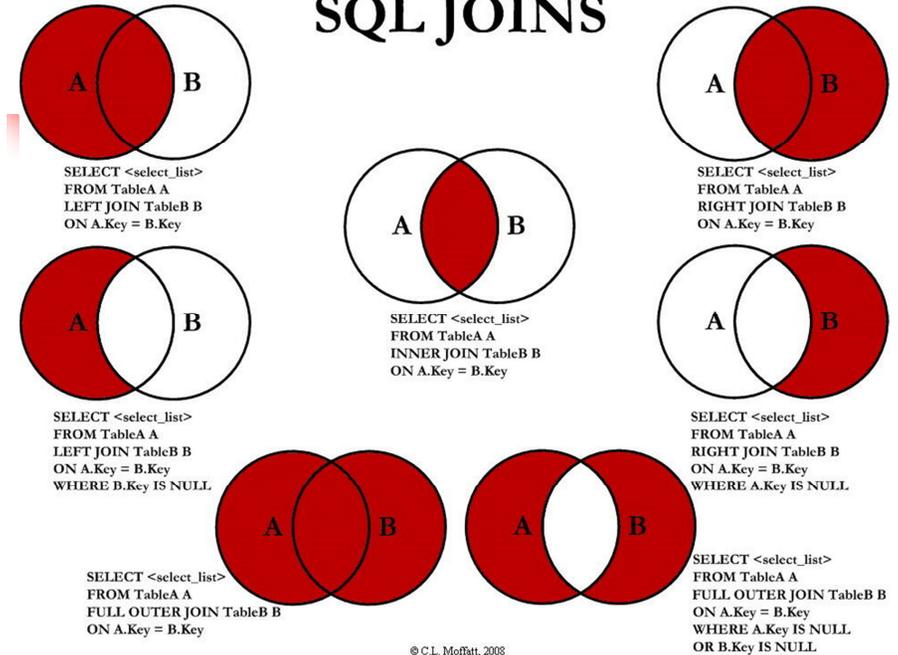
40

## Outerjoins

- **R OUTER JOIN S** incorpora además las tuplas que no satisfacen la condición (dangling tuples) o que tienen valores null.
- Sus variantes son:
  1. Opcional **NATURAL** delante de **OUTER**.
  2. Opcional **ON <condición>** luego de **JOIN**.
  3. Opcional **LEFT, RIGHT, o FULL** antes de **OUTER**.
    - ◆ **LEFT** = agrega sólo dangling tuples de R.
    - ◆ **RIGHT** = agrega sólo dangling tuples de S.
    - ◆ **FULL** = agrega dangling tuples R y S. Por defecto.

41

## SQL JOINS



## Operadores de Agregación

- **AVG (atributo)** o promedio. Se aplica sobre columnas numéricas.
- **COUNT (atributo)** retorna el número de valores no nulos.
- **MAX (atributo)** retorna el valor máximo.
- **MIN (atributo)** retorna el valor mínimo.
- **SUM (atributo)** retorna la suma.

43

## Operadores de Agregación

- **COUNT(\*)** retorna el número de tuplas de la relación.
- Se pueden combinar con **DISTINCT** u **ALL** (por defecto).
  - **COUNT(DISTINCT nombres)**
- No se pueden componer funciones agregadas, por ej. **max(avg())**.

44

## Ejemplos de Agregación

- Promedio de duración de las películas a color.

```
SELECT AVG(duracion)
FROM PELICULA
WHERE encolor;
```

- Total de tuplas en ACTUA.

```
SELECT COUNT(*)
FROM ACTUA;
```

- Cantidad actores distintos en ACTUA.

```
SELECT COUNT(DISTINCT actornombre)
FROM ACTUA;
```

45

## Agrupamiento

- Existen situaciones donde sería deseable aplicar la funciones agregadas a grupos de conjuntos de tuplas.
- En estas situaciones de usa la cláusula **GROUP BY** <lista atributos>.

PELICULA(titulo, anio, duracion, encolor, estudio, director)

```
SELECT estudio, AVG(duracion), MAX(anio) - MIN(anio)
FROM PELICULA
GROUP BY estudio;
```

46

## Restricciones de la Agregación

- Si en la consulta se utiliza alguna función agregada, entonces cada elemento de la lista del SELECT debe ser:
  - Una función agregada, o
  - Un atributo presente en la lista de atributos del GROUP BY.

47

## Cláusula HAVING

- La cláusula WHERE aplica la selección sobre las tuplas, si quisieramos aplicar una selección de alguna condición sobre los grupos formados debemos utilizar la cláusula **HAVING**.
- **HAVING** <condición> puede seguir a una cláusula GROUP BY. Los grupos que no satisfacen la condición son eliminados.

48

## Ejemplo HAVING

- Supongamos querer conocer tiempo total filmado por director, pero solo de aquellos que hayan dirigido al menos una película antes del año 2000.

PELICULA(titulo, anio, duracion, encolor, estudio, director)

DIRECTOR(nombre, direccion, cert)

```
SELECT nombre, SUM(duracion)
FROM DIRECTOR, PELICULA
WHERE cert = director
GROUP BY nombre
HAVING MIN(anio) < 2000;
```

*¿Qué sucede si en lugar de la condición en el HAVING se hubiese puesto  
WHERE anio < 2000 and cert = director?*

49

## Resumen

- SELECT <lista de atributos y agregados>
- FROM <lista de relaciones>
- WHERE <condición sobre los atributos de las relaciones>
- GROUP BY <lista de atributos>
- HAVING <condiciones sobre los grupos>
- ORDER BY <lista de atributos>

50

## Valores Null

- Null es un valor especial que puede tener un atributo.
- Puede interpretarse de diversas formas:
  - Valor desconocido: No disponible por el momento. Por ej. Si se desconoce la fecha de nacimiento.
  - Valor inaplicable: Ningún valor tiene sentido. Ej. Un atributo " nombre cónyuge " para un soltero/a.
  - Valor retenido: No se dispone de la suficiente jerarquía para conocer el valor. Por ej. Un número de teléfono, pin, etc.

51

## Aritmética de Valores Null

- Las condiciones en la cláusula WHERE deben estar preparadas para operar con valores NULL.
- Reglas para operar con valores NULL:
  - Operadores aritmeticos (+,\*,etc.): Cuando al menos uno de los operandos es null, el resultado de la operación es null.
  - Operadores relacionales (<,>,,=,etc.): Cuando al menos uno de los operandos es null, el resultado de la operación es unknown.

52

## Aritmética de Valores Null

- El valor Null no es una constante, es decir, no puede ser utilizado explícitamente como un operando.
  - Si x tiene valor NULL:
    - x + 3 es NULL
    - NULL + 3 no es válido
    - x = 3 es UNKNOWN
    - NULL = 3 no es válido
- Para conocer si un cierto atributo x tiene valor null se utiliza la expresión "x IS NULL" o "x IS NOT NULL"

53

## Valor de Verdad de UNKNOWN

UNKNOWN TRUE FALSE

- Para las comparaciones se utiliza una lógica de 3 valores (*true*, *false* y *unknown*) propuesta por CODD.

x	y	x AND y	x OR y	NOT x	x = y
TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE	UNKNOWN
TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE	UNKNOWN
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE

54

## Valor de Verdad de UNKNOWN

- Una regla sencilla para recordar la tabla anterior es utilizar las siguientes asignaciones, *true=1*, *false=0* y *unknown=1/2*.
  - AND: el resultado es el *mínimo* de los operandos.
  - OR: el resultado es el *máximo* de los operandos.
  - NOT: el resultado es *1-v* donde v es el valor del operando.
- Las sentencias SQL **solo mostrarán aquellas tuplas que produzcan un valor de verdad true** como resultado de la condición en la cláusula where.

55

## Observaciones Unknown y Null

PELICULA(titulo, anio, duracion, encolor, estudio, director)

```
SELECT titulo, duracion
FROM PELICULA
WHERE duracion <= "03:00:00" OR duracion > "03:00:00";
```

- La consulta anterior retorna **todas las tuplas** de la relación **que no tengan** en el atributo duracion un valor **null** (ya que sino produciría un valor *unknown* y esta tupla no forma parte de la respuesta).

56

## Funciones agregadas con Null.

- Los valores NULL son ignorados por cualquier función de agregación con excepción de **count(\*)**.

Ejnull	
i	j
150	150
200	200
350	350
null	0

```
SELECT COUNT(*)
FROM EJNULL;
```

Resultado: 4

```
SELECT COUNT(i)
FROM EJNULL;
```

Resultado: 3

```
SELECT AVG(i)
FROM EJNULL;
```

Resultado: 233,333

```
SELECT AVG(j)
FROM EJNULL;
```

Resultado: 175,000

57

## Agrupamientos con valores Null.

- Los valores NULL son tratados como un valor convencional en los agrupamientos por atributos.

Ejnull		
i	j	k
150	150	a
200	200	b
350	350	null
null	0	null

```
SELECT k, COUNT(k), COUNT(*), AVG(i), AVG(j)
FROM EJNULL
GROUP BY k;
```

Resultado:

k	count(k)	count(*)	AVG(i)	AVG(j)
NULL	0	2	350	175
a	1	1	150	150
b	1	1	200	200

58

## Actualizando el contenido de la Base de Datos

*Las siguientes instrucciones también son parte del DML y sirven para actualizar (agregar, borrar o modificar) el contenido de las relaciones.*

59

## Agregar filas

- Existen varias formas de agregar datos a una tabla. Sea,
  - $R$  representa una relación
  - $A_i$  representa un atributo
  - $V_i$  representa un valor.

1) **INSERT INTO R**  
**VALUES** ( $V_1, \dots, V_n$ )

*ACTOR(nombre, direccion, sexo, fechanac)*

```
INSERT INTO ACTOR
VALUES ('Leonard Nimoy', '6th Av. 600', 'M', NULL);
```

60

## Agregar filas

- Se pueden especificar los atributos:
  - para ingresar los datos en un orden particular, o
  - para que queden documentado los insert.

2) **INSERT INTO**  $R(A_1, \dots, A_n)$   
**VALUES**  $(V_1, \dots, V_n)$

```
INSERT INTO ACTOR(direccion, fechanac, nombre, sexo)
VALUES ('6th Av. 600', NULL, 'Winona Ryder', 'F');
```

61

## Agregar filas

- Se pueden especificar sólo algunos atributos en cualquier orden y los demás toman el valor por defecto de cada tipo.

3) **INSERT INTO**  $R(A_1, \dots, A_m)$   
**VALUES**  $(V_1, \dots, V_m)$

- $m < n$

*PELICULA(titulo, anio, duracion, encolor, estudio, director)*

```
INSERT INTO PELICULA(anio, titulo, director, estudio)
VALUES (2003, 'Terminator 3: Rise of the Machines', 103, 3);
```

62

## Agregar filas

- Las alternativas anteriores sólo insertan una tupla en la relación.
- Podemos insertar múltiples tuplas en una relación en una sola sentencia utilizando consultas.

*ACTORDIRECTOR(nombre, sexo, cantdir)*

```
INSERT INTO ACTORDIRECTOR(nombre, cantdir)
SELECT nombre, count(*) as cantdir
FROM PELICULA JOIN DIRECTOR ON director = cert
WHERE DIRECTOR.nombre IN (SELECT actornombre FROM ACTUA)
GROUP BY DIRECTOR.cert;
```

63

## Borrar Filas de una tabla

- Borrar todas las tuplas de una tabla:

**DELETE FROM**  $R$ ;

```
DELETE FROM actordirector;
```

- Para borrar algunas tuplas es necesario describirlas.

**DELETE FROM**  $R$   
**WHERE** *<condición>*;

```
DELETE FROM actordirector;
```

64

## Observaciones sobre DELETE

- La operación DELETE sin cláusula de condición borra todas las filas de una tabla, **no borra su esquema**.
- La operación de DELETE será exitosa siempre que no se violen las restricciones preexistentes.

65

## Actualizar filas de una tabla

- Actualizar todas las tuplas de una tabla:

**UPDATE R SET** <nuevas asignaciones>;

```
UPDATE ACTOR SET fechanac = "1990-01-01";
```

- Para actualizar algunas tuplas es necesario describirlas.

**UPDATE R SET** <nuevas asignaciones>;

**WHERE** <condición>;

```
UPDATE ACTOR SET sexo = 'F'  
WHERE sexo IS NULL;
```

66

## Observaciones sobre UPDATE

- Al igual que en el caso de INSERT y DELETE esta operación solo será exitosa si como resultado de la modificación se siguen respetando todas las restricciones preexistentes.

67

## Resumen

- DML sobre el contenido de las tablas:
  - Agregar una o más filas INSERT INTO
  - Borrar una o más filas DELETE FROM
  - Actualizar el contenido de una o más filas UPDATE ... SET

68

## Lenguaje de Definición de Datos

*Estas operaciones sólo está autorizado a realizarla el DBA o administrador de la base de datos*

69

## Instrucciones del LDD

- Para la definición de esquemas:

```
CREATE TABLE tab_nombre (  
    atrib1 TIPO [<restricción>],  
    atrib2 TIPO [<restricción>],  
    ....  
    <restriccion_integridad_1>,  
    ....  
    <restriccion_integridad_n>);
```

70

## Creación de Esquemas

- Definir el esquema de la tabla (atributos y sus tipos).
- Definir, si existen, restricciones sobre los atributos.
  - Condición de no nulo, restricciones de valores, subconjunto válido, etc.
- Definir restricciones a nivel de tabla
  - Clave primaria, clave foránea, índices, restricciones entre columnas.

- *Ejemplo:*

```
create table actordirector  
    (nombre    varchar(40) not null,  
    sexo      enum ('M','F'),  
    cantdir   integer unsigned)
```

71

## Restricciones de Integridad

- **not null**
- **primary key** ( $A_1, \dots, A_n$ )
- **foreign key** ( $A_1, \dots, A_n$ ) **references** R( $B_1, \dots, B_n$ )

```
CREATE TABLE ACTORDIRECTOR2(  
    nombre VARCHAR(40) NOT NULL,  
    sexo ENUM ('M','F'),  
    cantdir INT UNSIGNED,  
    PRIMARY KEY(nombre),  
    FOREIGN KEY(nombre) REFERENCES ACTUA(actornombre)  
) ENGINE=InnoDB;
```

72

## Instrucción ALTER TABLE

- Permite modificar definiciones del esquema de una tabla.
- En líneas generales la definición de las tablas debe ser estática, una vez creado todo el esquema para la base de datos se espera no tener que modificarlo.

```
ALTER TABLE <nombre_tabla>  
  DROP columna1,  
  MODIFY columna2 <modificación>,  
  ADD columna3 <tipo> [<restricción>],  
  ADD/DROP CONSTRAINT restricción1 ...
```

73

## Borrado de Esquemas

- La instrucción DROP TABLE permite borrar el contenido y el esquema de una tabla.
- Para poder borrar una tabla la misma no debe estar referenciada.

74

## Temas de la Clase de Hoy

- Lenguaje de Consulta Relacional Formal
  - Calculo Relacional de Tuplas (CRT)
    - Estructura general.
    - Átomos y Fórmulas.
    - Fórmulas Seguras.
- Lenguajes Consulta Relacional Comercial
  - SQL
    - Instrucción SELECT/FROM/WHERE.
    - Instrucciones de actualización de contenido.
    - Instrucciones de DLL.

75

## Bibliografía

- Bibliografía
  - "Fundamentos de Bases de Datos" – A. Silberschatz. Capítulo 4.
  - "Database and Knowledge Base System" – J. Ullman. Capítulo 6.
  - <http://dev.mysql.com/doc/refman/5.4/en/>

### Referencias

- <http://www.w3schools.com/sql/>

76