SQL: Structured Query Language

M. Andrea Rodríguez-Tastets

Universidad de Concepción,Chile
www.inf.udec.cl\ ∼andrea
andrea@udec.cl

II Semestre - 2014

Introducción

Manipulación de Tablas

Especificación de Restricciones

Consultas Básicas

Subconsultas

Null y Unknown

Restricciones a nivel de esquema y triggers

Views

Indices en SQL

Recursividad en SQL

Objetivos de la Unidad

Estudiar SQL y su relación con otros lenguajes de manipulación en un modelo relacional

Introducción

- SQL (Structured Query Language)
- SQL emplea los términos tabla, fila y columna en vez de relación, tupla y atributo, respectivamente.
- Las instrucciones SQL para definir datos son CREATE, ALTER y DROP

Evolution

Existen los siguientes estándares:

- ANSI (American National Standard Institute) SQL con sus actualizaciones en 1992, llamado SQL2. Más reciente es el SQL-1999 (SQL3) que extiende el SQL2 con características objecto-relacional y otras funcionalidades.
- Existe una colección de extensiones al SQL:99 llamadas SQL:2003 que incluyen, por ejemplo soporte para XML.
- También hay versiones de SQL producidas por vendedores de DBMS, las cuales cumplen con las propiedades del SQL original y el SQL2, tendiendo variaciones en cuanto al SQL3 y el SQL:2003.

Concepto de Esquema y Catálogo

- Un esquema SQL se identifica con un nombre de esquema y consta de un identificador de autorización, que indica el usuario o la cuenta propietaria del esquema, además de los descriptores de cada elemento del esquema.
- Los elementos del esquema comprenden tablas, restricciones, vistas, dominios y otros.
- Un esquema se crea mediante la sentencia CREATE SCHEMA, por ejemplo:

  CREATE SCHEMA EMPRESA AUTHORIZATION JPEREZ;
- Catálogo es un conjunto de esquemas, con un nombre. El catálogo, siempre contiene un esquema especial, llamado INFORMATION_SCHEMA, que proporciona información sobre todos los esquemas del catálogo.
- La ventaja de los catálogos es que se pueden definir restricciones de integridad, sobre relaciones que están en el mismo catálogo. Además los esquemas del mismo catálogo pueden compartir ciertos elementos, como definiciones de dominio.

CREATE TABLE

- La instrucción CREATE TABLE permite crear relaciones, dándole un nombre y especificando atributos y restricciones.
- Los atributos se especifican con un nombre, un tipo de datos (para especificar su dominio de valores) y cualquier restricción del mismo, por ejemplo: NOT NULL.
- Se especifican las restricciones de clave: de integridad de la entidad y referencial

```
CREATE TABLE    EMPLEADO
(NOMBRE         VARCHAR(15) NOT NULL,
INIC            CHAR,
APELLIDO        VARCHAR(15) NOT NULL,
NUMERO          CHAR(9) NOT NULL,
FECHA_NAC       DATE,
DIRECCION       VARCHAR(30),
SEXO            CHAR,
SALARIO         DECIMAL(10,2),
NRO_SUP         CHAR(9),
NRO_DEPTO       INT NOT NULL,
PRIMARY KEY     (NUMERO),
FOREIGN KEY     (NRO_SUP) REFERENCES EMPLEADO (NUMERO),
FOREIGN KEY     (NRO_DEPTO) REFERENCES DEPARTAMENTO (ND));
```

Tipos de Datos y Dominios (1/2)

- Tipos de Datos para atributos: numéricos, cadena de caracteres, cadena de bits, fecha y hora.
- Numéricos (de distintos tamaños, enteros): INTEGER o INT y SMALLINT y reales de diversas precisiones (FLOAT; REAL; DOUBLE PRECISION). Formatos DECIMAL(i,j) o DEC(i,j) o NUMERIC(i,j), donde i es la precisión (nro de dígitos) y j la escala (número de decimales)
- Cadena de caracteres, de longitud fija (CHAR(n) o CHARACTER(n), donde *n* es el número de caracteres) o de longitud variable (VARCHAR(n) o CHARVARYING(n), donde *n* es el número máximo de caracteres)
- Cadena de bits, pueden ser de longitud fija n (BIT(n)) o longitud variable (BIT VARIYNG(n)), donde *n* es el número máximo de bits.

Tipos de Datos y Dominios (2/2)

- Tipo de dato para fecha es DATE, tiene 10 posiciones y sus componentes son YEAR, MONTH y DAY, por lo regular de la forma YYYY-MM-DD.
- Tipo de dato para tiempo es TIME, tiene 8 posiciones, con los componentes HOUR, MINUTE y SECOND, por lo general de la forma HH:MM:SS
- Dominios: esto permite cambiar tipos de datos de un dominio utilizado por un gran número de atributos más fácilmente. Por ejemplo, dominio TIPO_NUMERO así: CREATE DOMAIN TIPO_NUMERO AS CHAR(9)

DROP TABLE

- Existen dos opciones para eliminar: CASCADE y RESTRICT.
- Si se desea eliminar el esquema EMPRESA con todas sus tablas, dominios, etc., se utiliza CASCADE: DROP SCHEMA EMPRESA CASCADE;
- Si se elige la opción RESTRICT, el esquema se eliminará sólo si no tiene elementos; en caso contrario no se ejecutará la instrucción DROP
- DROP TABLE DEPENDIENTE CASCADE; Elimina la tabla DEPENDIENTE. Con la opción CASCADE, todas las restricciones y vistas que hagan referencia a la tabla se eliminarán automáticamente del esquema, junto con la propia tabla.

ALTER TABLE

- Las posibles acciones de alterar tablas incluyen agregar atributos, eliminar atributos, la modificación de la definición de una columna y la agregación y eliminación de restricciones de tabla.
- ALTER TABLE EMPRESA.EMPLEADO ADD PUESTO VARCHAR(12); Agrega un atributo. Para darle valores a este nuevo atributo se puede utilizar UPDATE o bien darle un valor por defecto. Si no se hace esto último, el nuevo atributo tendrá valor NULL
- ALTER TABLE EMPRESA.EMPLEADO DROP DIRECCION CASCADE; Elimina atributo DIRECCION de la tabla EMPLEADO

Especificación de Restricciones (1/4)

- Dado que SQL permite NULL como valor para un atributo, se puede especificar la restricción NOT NULL, si es que no queremos nulos.
- Valor por omisión de un atributo, añadiendo la cláusula DEFAULT <valor> a la definición de un atributo
- La cláusula PRIMARY KEY y FOREIGN KEY permiten especificar las PK y FK.
- La cláusula UNIQUE especifica claves alternativas
- El diseñador debe especificar las acciones que deben realizarse cuando se transgrede una restricción, asociada a la inserción, modificación y eliminación de un valor de atributo de clave foránea y asociada al modificarse un valor de clave primaria referenciada, añadiendo una cláusula de acción de disparo referencial a una restricción de clave externa.
- Las opciones son: SET NULL, CASCADE y SET DEFAULT, las opciones son ON DELETE o ON UPDATE.

Especificación de Restricciones (2/4)

```
CREATE TABLE    EMPLEADO
(...,
NRO_DEPTO       INT NOT NULL DEFAULT 1,
PRIMARY KEY     (NUMERO),
FOREIGN KEY     (NRO_SUP) REFERENCES EMPLEADO (NUMERO)
                ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY     (NRO_DEPTO) REFERENCES DEPARTAMENTO (ND)
                ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

Especificación de Restricciones (3/4)

- Se tiene SET NULL ON DELETE y CASCADE ON UPDATE para la clave externa NRO_SUP de EMPLEADO. Esto significa que si se elimina la tupla de empleado supervisor, el valor de NRO_SUP pasa automáticamente a NULL en todas las tuplas de EMPLEADO que hagan referencia a la tupla eliminada.
- Si se actualiza el valor de NUMERO de un empleado supervisor, el nuevo valor se propaga a NRO_SUP para todas las tuplas de empleado que hagan referencia a la tupla del empleado actualizada.
- En general, la acción emprendida por el SGBD cuando se especifica SET NULL o SET DEFAULT es la misma para ON DELETE que para ON UPDATE: el valor de los atributos referenciados afectados se cambia a NULL en el caso de SET NULL y al valor por omisión especificado en el caso de SET DEFAULT.

Especificación de Restricciones (4/4)

- La acción correspondiente a CASCADE ON DELETE es eliminar todas las tuplas referenciadoras, mientras que la acción correspondiente a CASCADE ON UPDATE es cambiar el valor de la clave externa al nuevo valor actualizado de la clave primaria en todas las tuplas referenciadoras.
- Por lo general, la opción CASCADE es adecuada para tablas del tipo TRABAJA_EN, para relaciones que representan atributos multivaluados como LOCALIZACION_DEPTO y para relaciones que representan tipos de entidad débiles como DEPENDIENTE.
- Las relaciones declaradas con CREATE TABLE se denominan tablas base. El SGBD las almacena como archivos.
- En SQL, los atributos de una relación de una tabla base están ordenados en la secuencia en que se especifican en la sentencia CREATE TABLE. Las filas no están ordenadas.

Consultas Básicas

- La sentencia SELECT permite recuperar información de la BD. No tiene nada que ver con la operación de selección del álgebra relacional.
- SQL y el Modelo relacional tienen diferencias: SQL permite filas duplicadas, es decir, una tabla SQL no es un conjunto de tuplas, porque los conjuntos no permiten elementos repetidos. A veces a las tablas SQL se les llama bolsa, bag o multiconjunto.
- La cláusula DISTINCT elimina las filas repetidas.

SELECT-FROM-WHERE

SELECT <lista de atributos>
- FROM <lista de tablas>

WHERE <condición>

- Considere: Recupere la fecha de nacimiento y direccin del empleado cuyo nombre es Juan Pérez.

```
SELECT   FECHA_NAC, DIRECCION
FROM     EMPLEADO
WHERE    Nombre='Juan' AND APELLIDO = 'Pérez';
```

Ejercicio

EMPLEADO	(Nombre, Apellido, NroEmpleado, FechaNacimiento, Direccion, Sexo, Salario, NroSuperior, NroDepto) FK NroDepto de DEPARTAMENTO FK NroSupervisor de EMPLEADO
DEPARTAMENTO	(NombreDepto, NroDepto, NroGerente, FechaInicioGerente) FK NroGerente de EMPLEADO
LUGARESDEPTO	(NroDepto, LugarDepto) FK NroDepto de DEPARTAMENTO
PROYECTO	(NombreProyecto, NroProyecto, LugarProyecto, NroDepto) FK NroDepto de DEPARTMENTO
TRABAJAEN	(NroEmpleado, NroProyecto, Horas) FK NroEmpleado de EMPLEADO, FK NroProyecto de PROYECTO
DEPENDIENTE	(NroEmpleado, NombreDependiente, Sexo, FechaNacimiento, Parentesco) FK NroEmpleado de EMPLEADO

Consultas 1

Recupere el nombre, apellido y dirección de todos los empleados que trabajan en el departamento de Investigación

```
SELECT    E.NOMBRE, E.APELLIDO, E.DIRECCION
FROM      EMPLEADO AS E, DEPARTAMENTO
WHERE     DEPARTAMENTO.NOMBREDEPTO = 'Investigación' AND
          DEPARTAMENTO.NRODEPTO = E.NRODEPTO;
```

Equivalente:

```
SELECT    NOMBRE, APELLIDO, DIRECCION
FROM      EMPLEADO INNER JOIN DEPARTAMENTO ON
          (DEPARTAMENTO.NRODEPTO = EMPLEADO.NRODEPTO AND
          DEPARTAMENTO.NOMBREDEPTO = 'Investigación');
```

Consultas 2

De cada proyecto ubicado en Concepción, haga una lista con el número de proyecto, el número de departamento controlador y el apellido, dirección y fecha de nacimiento del jefe de departamento

```
SELECT    NROPROYECTO, NRODEPTO, APELLIDO, DIRECCION, FECHANACIMIENTO
FROM      EMPLEADO, DEPARTAMENTO, PROYECTO
WHERE     PROYECTO.NRODEPTO = DEPARTAMENTO.NRODEPTO AND
          DEPARTAMENTO.NROGERENTE = EMPLEADO.NROEMPLEADO
          AND LUGARPROYECTO = Concepción;
```

Consultas sin WHERE

Seleccione todos los números de empleado de EMPLEADO

```
SELECT    NROEMPLEADO
FROM      EMPLEADO;
```

Seleccione todas las combinaciones posibles de números de empleado y números de departamento (Esta consulta realiza el producto cartesiano).

```
SELECT    NROEMPLEADO, NRODEPARTAMENTO
FROM      EMPLEADO, DEPARTAMENTO;
```

Uso del *

Uso del $*$: Recupera los valores de todos los atributos

```
SELECT    *
FROM      EMPLEADO
WHERE     NroDepto = 5;
```

Tablas como Conjuntos en SQL

- SQL no elimina las filas repetidas en los resultados de las consultas, por las siguientes razones:
  - La eliminación de duplicados es una operación costosa. Una forma de implementarla es ordenar las tuplas primero y luego eliminar los duplicados.
  - Es posible que el usuario requiera ver las tuplas repetidas en el resultado de la consulta
  - Cuando se aplica una función agregada, en la mayoría de los casos no se quiere eliminar los duplicados.
- Si queremos eliminar tuplas repetidas en el resultado de una consulta, se usa la palabra clave DISTINCT en la cláusula SELECT. Por ejemplo: Recupere el salario de todos los empleados y los valores de todos los salarios distintos

```
SELECT    ALL SALARIO (o SELECT simple)
FROM      EMPLEADO; (incluye repetidos)
SELECT    DISTINCT SALARIO
FROM      EMPLEADO; (no incluye repetidos)
```

UNION, EXCEPT e INTERSECT

UNION, EXCEPT e INTERSECT, equivalentes a la unión, diferencia e intersección de conjuntos. Las tuplas repetidas se eliminan del resultado y se debe asegurar que las relaciones sean compatibles en su esquema (mismos atributos y el orden). Por ejemplo, liste con todos los números de proyecto en los que participa un empleado de apellido Pérez, sea como trabajador o como jefe del departamento que controla el proyecto.

```
(SELECT    DISTINCT NROPROYECTO
FROM       PROYECTO, DEPARTAMENTO, EMPLEADO
WHERE      PROYECTO.NRODEPTO = DEPARTAMENTO.NRODEPTO AND
           DEPARTAMENTO.NROGERENTE = EMPLEADO.NROEMPLEADO
           AND APELLIDO = 'Pérez')
UNION
(SELECT    DISTINCT NROPROYECTO
FROM       TRABAJA_EN, EMPLEADO
WHERE      TRABAJA_EN. NROEMPLEADO = EMPLEADO.NROEMPLEADO
           AND APELLIDO = 'Pérez');
```

Comparación y ordenación

Comparación cadena de caracteres: operador LIKE, cadenas parciales %: sustituye un número arbitrareo de caracteres y _ sustituye a un solo caracter. Por ejemplo, Recupere todos los empleados cuya dirección sea Talcahuano.

```
SELECT    NOMBRE, APELLIDO
FROM      EMPLEADO
WHERE     DIRECCION LIKE '% Talcahuano';
```

Aritmética en Consultas

Los operadores suma (+), resta (−), multiplicación (∗) y división (/) se pueden aplicar a valores numéricos o atributos con dominios númericos. Por ejemplo, Muestre los salarios resultantes si cada empleado que trabaja en el proyecto 'Producto X' recibe un aumento del 10 %.

```
SELECT    NOMBRE, APELLIDO, 1.1*SALARIO
FROM      EMPLEADO, TRABAJA_EN, PROYECTO
WHERE     TRABAJA_EN. NROEMPLEADO = EMPLEADO.NROEMPLEADO AND
          TRABAJA_EN.NROPROYECTO = PROYECTO.NROPROYECTO AND
          NOMBREPROYECTO = 'Producto X';
```

En el caso de tipos de datos de cadena, se puede usar el operador concatenación(//) en una consulta para anexar un valor de cadena a otro. En el caso de tipos de datos fecha, tiempo, los operadores incluyen el incremento (+) o disminución (−), en un intervalo compatible con el tipo de una fecha, o tiempo.

Operador BETWEEN

Recupere todos los empleados del departamento 5 cuyo salario esté entre 30000 y 40000 dólares.

```
SELECT    *
FROM      EMPLEADO
WHERE     (SALARIO BETWEEN 30000 AND 40000) AND NRODEPTO= 5;
```

Cláusula ORDER BY

Obtenga una lista de los empleados y de los proyectos en los que trabajan, ordenados por departamento, y dentro de cada departamento, alfabéticamente por apellido y nombre.

```
SELECT   NOMBREDEPTO, APELLIDO, NOMBRE, NOMBREPROYECTO
FROM     DEPARTAMENTO, EMPLEADO, TRABAJA_EN, PROYECTO
WHERE    DEPARTAMENTO.NRODEPTO = EMPLEADO.NRODEPTO AND
         EMPLEADO.NROEMPLEADO = TRABAJA_EN.NROEMPLEADO AND
         TRABAJA_EN.NRO.PROY = PROYECTO.NROPROY
         ORDER BY NOMBREDEPTO, APELLIDO, NOMBRE;
```

El orden por omisión es ascendente. Se usa la palabra DESC para ordenar en forma descendente. Por ejemplo, ORDER BY NOMBREDEPTO DESC, APELLIDO ASC, NOMBRE ASC

Conjuntos Explícitos y NULL

Recupere el número de empleado de todos los que trabajan en los proyectos 1, 2 o 3.

```
SELECT DISTINCT NROEMPLEADO
FROM EMPLEADO
WHERE NROPROY IN (1,2,3);
```

Recupere los nombres de todos los empleados que no tienen supervisores.

```
SELECT NOMBRE, APELLIDO
FROM EMPLEADO
WHERE NROSUPERVISOR IS NULL;
```

Subconsultas en el WHERE

- Estas consultas requieren valores de la BD para usarlos después en una condición de comparación.
- Consultas anidadas: bloques select-from-where dentro de la cláusula where de otra consulta. A esta última consulta se le llama consulta externa.

```
SELECT DISTINCT NROPROY
FROM PROYECTO
WHERE NROPROY IN        ( SELECT NROPROY
                          FROM PROYECTO, EMPLEADO
                          WHERE PROYECTO.NRODEPTO=DEPTO. NRODEPTO AND
                          NRO NROGERENTE = NROSUPERIOR AND APELLIDO ='Pérez')
OR NROPROY IN           (SELECT NROPROY
                          FROM TRABAJA_EN, EMPLEADO
                          WHERE EMPLEADO.NROEMP = TRABAJA_EN.NROEMP
                          AND APELLIDO = 'Pérez');
```

Consultas anidadas (WHERE)

```
SELECT DISTINCT NROPROY
FROM PROYECTO
WHERE NROPROY IN        ( SELECT NROPROY
                          FROM PROYECTO, EMPLEADO
                          WHERE PROYECTO.NRODEPTO=DEPTO. NRODEPTO AND
                          NRO NROGERENTE = NROSUPERIOR AND APELLIDO ='Pérez')
OR NROPROY IN           (SELECT NROPROY
                          FROM TRABAJA_EN, EMPLEADO
                          WHERE EMPLEADO.NROEMP = TRABAJA_EN.NROEMP
                          AND APELLIDO = 'Pérez');
```

- La primera consulta anidada selecciona los números de proyectos en que un Pérez participa como jefe.
- La segunda consulta anidada selecciona los números de proyectos en que un Pérez participa como trabajador.
- La consulta externa selecciona una tupla PROYECTO si el valor de NROPROY de esa tupla está en el resultado de cualquiera de las dos consultas anidadas.
- El operador de comparación IN compara el valor v con un conjunto (o multiconjunto) de valores V y evalúa a TRUE para comprobar si v es uno de los elementos de V

Consultas Anidadas Correlacionadas

- Siempre que una condición en la cláusula WHERE de una consulta anidada hace referencia a un atributo de una relación declarada en la consulta externa, se dice que las dos consultas están correlacionadas.
- La consulta anidada se evalúa una sola vez para cada tupla (o combinación de tuplas) en la consulta externa
- Ejemplo: Recupere el nombre de cada empleado que tenga un familiar dependiente con el mismo nombre de pila y sexo que el empleado.

```
SELECT    E.NOMBRE, E.APELLIDO
FROM      EMPLEADO AS E, DEPENDIENTE AS D
WHERE     E.NROEMPLEADO =D.NROEMPLEADO AND
          E.SEXO =D.SEXO AND E.NOMBRE =D.NOMBREDEP;
```

EXISTS

EXISTS sirve para comprobar si el resultado de una consulta anidada correlacionada es o no vacío.

```
SELECT E.NOMBRE, E.APELLIDO
FROM EMPLEADO AS E
WHERE EXISTS ( SELECT *
                    FROM DEPENDIENTE
                    WHERE E.NROEMP = DEPENDIENTE.NROEMP
                    AND SEXO = E.SEXO AND E.NOMBRE =NOMBREDEPENDIENTE);
```

Recupere los nombres de empleados que no tienen familiares dependientes

```
SELECT NOMBRE, APELLIDO
FROM EMPLEADO
WHERE NOT EXISTS (SELECT *
                    FROM DEPENDIENTE
                    WHERE EMPLEADO.NROEMPLEADO = DEPENDIENTE.NROEMPLEADO);
```

Subconsultas en el FROM

Encuentre el nombre de los empleados que trabajan en Departamento que tienen proyectos en Concepción.

```
SELECT NOMBRE
FROM EMPLEADO AS E,
        (SELECT NroDEPTO FROM PROYECTO
        WHERE Lugar= 'Concepción') AS PRO
WHERE E.NroDepto = PRO.NroDepto);
```

Funciones agregadas y agrupación (1/2)

Funciones COUNT (cuenta número de tuplas), SUM, MAX, MIN, AVG. Por ejemplo, Halle la suma de los salarios de todos los empleados, el salario máximo, el mínimo y el salario medio.

```
SELECT    SUM(SALARIO), MAX(SALARIO), MIN(SALARIO), AVG(SALARIO)
FROM      EMPLEADO;
```

Recupere el total de empleados de la empresa

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

Funciones agregadas y agrupación (2/2)

Recupere el número de empleados del departamento de Investigación

```
SELECT    COUNT(*)
FROM      EMPLEADO, DEPARTAMENTO
WHERE     DEPTO.NRODEPTO = EMPLEADO.NRODEPTO AND
          NOMBREDEPTO =Investigacin;
```

Contar valores de atributos en vez de tuplas

```
SELECT    COUNT(DISTINCT SALARIO)
FROM      EMPLEADO;
```

Agrupación (1/4)

- A veces es necesario aplicar funciones agregadas a subgrupos de tuplas de una relación, por ejemplo conocer el salario medio de los empleados de cada departamento, o el número de empleados que trabajan en que cada proyecto.
- En estos casos se necesita agrupar las tuplas que tienen el mismo valor para ciertos atributos, que se llaman atributos de agrupación y aplicar la función de manera independiente a cada uno de esos grupos.
- SQL tiene la cláusula GROUP BY

Agrupación (2/4)

Recupere el número de depto y el número de empleados de cada departamento y su salario medio.

```
SELECT    NRODEPTO, COUNT(*), AVG (SALARIO)
FROM      EMPLEADO GROUP BY NRODEPTO;
```

De cada proyecto, recupere su número, nombre y el número de empleados que trabajan en él

```
SELECT      NROPROY, NOMBREPROY, COUNT(*)
FROM        PROYECTO, TRABAJA_EN
WHERE       PROYECTO.NROPROY = TRABAJA_EN.NROPROY
GROUP BY    NROPROY, NOMBREPROY;
```

Agrupación (3/4)

- A veces se requiere recuperar valores agrupados cuando se cumplan ciertas condiciones. Por ejemplo, la consulta anterior, pero para aquellos proyectos que tengan más de dos empleados.
- Para estos casos se ocupa la cláusula HAVING, que puede aparecer con la cláusula GROUP BY.
- HAVING especifica una condición en términos del grupo de tuplas asociado a cada valor de los atributos de agrupación. Sólo lo grupos que satisfagan la condición entrarán en el resultado de la consulta.

Agrupación (4/4)

De cada proyecto con un número de empleados mayor que dos, recupere su número, nombre y el número de empleados que trabajan en él

```
SELECT      NROPROY, NOMBREPROY, COUNT(*)
FROM        PROYECTO, TRABAJA_EN
WHERE       PROYECTO.NROPROY = TRABAJA_EN.NROPROY
GROUP BY    NROPROY, NOMBREPROY
HAVING      COUNT(*) > 2;
```

Valores Nulos

SQL permite valores nulos (NULL). Sus interpretaciones son:

- Valor desconocido
- Valor inaplicable
- Valor restringido

En la clausuras del WHERE de una consulta, hay dos reglas que se aplican al evaluar condiciones sobre nulos:

- El resultado de una operación aritmética sobre nulos en nulos
- Cuando se hace una comparación sobre el NULL, el valor de verdad es desconocido (UNKNOWN).
- NULL no se puede ocupar explícitamente como un operando.

Valor de verdad de UNKNOWN

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

Null: Ejercicio

Considere la siguiente instancia de la relación PERSONA:

id	nombre	edad	grado
18	Rozas	20	3
23	Mella	23	6
22	Pinto	25	4
50	Bravo	18	null

Se quiere encontrar a la persona con el mayor grado cuya edad sea menor que 21

Null: Ejercicio (cont.)

Para la consulta deseada anterior, ¿Hacen las dos siguientes consultas la pregunta deseada? ¿Qué respuesta dan estas consultas?

```
SELECT   p1.nombre
FROM     PERSONA as p1
WHERE    NOT EXISTS (SELECT * FROM PERSONA as p2
         WHERE p2.edad < 21 AND p1.grado <= p2.grado)
```

```
SELECT   p1.nombre
FROM     PERSONA as p1
WHERE    p1.grado >=ANY (SELECT p2.grado FROM PERSONA as p2
         WHERE p2.edad < 21)
```

Restricciones a nivel de esquema

- Una "assertion"(declaración) es una expresión SQL de valor booleano que debe ser verdadera todo el tiempo.
- Un trigger (disparador) es un serie de acciones asociadas con ciertos eventos, tales como la inserción en una relación particular, y que son activadas cuando un evento ocurre.

Check constraints: atributos y tuplas

generoCHAR(1) CHECK (gender IN ('F','M'))

```
CREATE TABLE EstrellaCine(     nombre CHAR(30) PRIMARY KEY,
   direccion VARCHAR(255),
   genero CHAR(1),
   nacimiento DATE,
   CHECK( genero = 'F' OR nombre NOT LIKE ' Ms. %'));
```

Assertion (1/3)

```
CREATE ASSERTION    RESTRIC_SALARIO
CHECK(NOT EXISTS    (SELECT *
                    FROM EMPLEADO E, EMPLEADO M, DEPARTAMENTO D
                    WHERE E.SALARIO > M.SALARIO AND E.ND = D.NUMEROD
                    AND D.NSS_JEFE = M.NSS))
```

- Especifica cuándo se viola una restricción general, lo que implica el aborto de una actualización.
- Se puede usar CHECK es la creación de dominios.

Assertion: Comparison de restricciones (2/3)

Tipo de restricción	Dónde se declara	Cuándo se activa	Se garantiza
check:atributo	atributo	inserción en relaciones o actualización de atributos	No en subconsultas
check:tuplas	elementos BD	inserción de relación o actualización de tupla	No ensubconsultas
ckeck:assertion	elementos BD	cualquier cambio	Si

Assertion: Comparison de restricciones (3/3)

```
CREATE TABLE Estudios(     nombre CHAR(30) PRIMARY KEY,
   direccion VARCHAR(255),
   ejecutivoC INT REFERENCES Ejecutivos(id),     CHECK (ejecutivoC NOT IN (
     SELECT id FROM Ejecutivos
     WHERE acciones < 1000000)));


CREATE ASSERTION Presidentes(     (NOT EXISTS (
     (SELECT *
     FROM Estudios, Ejecutivos
     WHERE ejecutivoC = id AND acciones < 1000000));
```

Triggers(1/2)

Son disparadores de acciones ante eventos. Esto posibilita no que se aborte una actualización, sino que se avise o que realice una acción determinada. Esto último relacionado al concepto de bases de datos activas.

```
R1: CREATE TRIGGER              SALTOTAL1
AFTER INSERT ON                 EMPLEADO
FOR EACH ROW
WHEN (NEW.ND IS NOT NULL)
UPDATE                          DEPARTAMENTO SET SAL_TOTAL = SAL_TOTAL + NEW.SALARIO
WHERE                           ND = NEW.ND;
```

Triggers(2/2)

<disparador>::= CREATE TRIGGER <nombre de disparador>
(AFTER — BEFORE) <eventos de disparador> ON <nombre tabla>
[FOR EACH ROW]
[WHEN <condicion>]
<acciones del disparador>;
<eventos del disparador> ::= <evento del disparador>
{OR <evento del disparador>}
<evento del disparador> ::= INSERT | DELETE | UPDATE
[OF <nombre de columna> {, <nombre de columna>}]
<acciones del disparador> ::= <bloque PL/SQL>

Views

- Los usuarios que acceden a una base de datos relacional, lo hacen típicamente a través de vistas, de modo que diferentes usuarios tienen diferentes vistas.
- Una vista es una tabla virtual derivada, con nombre (Date, 1995). El término virtual significa que la tabla no existe en sí, pero para el usuario parece existir. Por el contrario una tabla base es real, en el sentido que existe almacenada en algún dispositivo físico de almacenamiento. Las vistas no se sustentan en datos almacenados, sólo se almacena su definición en el catálogo, en base a otras tablas.

Views:Ejemplo 1

```
CREATE VIEW TRABAJA_EN1
AS SELECT NOMBRE,APPELLIDO,NOMBREP, HORAS
FROM EMPLEADO,PROYECTO,TRABAJA_EN
WHERE NSS=NSSE AND NP = NUMEROP;
```

Views:Ejemplo 1

CREATE VIEW INFO_DEPTO(NOMBRED,NUM_EMP,SALA_TOTAL)
AS SELECT NOMBRED,COUNT(*), SUM(SALARIO)
FROM DEPARTAMENTO, EMPLEADO
WHERE ND = NUMEROPD
GROUP BY NOMBRED;

Consulta pueden ser especificadas sobre vistas.

Views:Ejemplo 3

CREATE VIEW TRABAJA_EN1
AS SELECT NOMBRE,APPELLIDO,NOMBREP,HORAS
FROM EMPLEADO, PROYECTO,TRABAJA_EN
WHERE NSS=NSSE AND NP = NUMEROP;

SELECT NOMBRE,APPELLIDO
FROM TRABAJA_EN1
WHERE NOMBREP = 'ProyectoX';

Implementación (1/2)

- Implementar eficientemente vistas es complejo.
- La estrategia de modificación de consultas conlleva covertir la consulta sobre la vista en una consulta sobre las tablas de base subyacentes.
- La modificación de consultas es una estrategia ineficiente para consultas complejas
- Otra estrategia es la materialización de vistas, la cual conlleva crear físicamente una tabla en el supuesto de que se realicen otras consultas sobre la vista.
- La materialización de vistas implica el desarrollar técnicas de actualización incremental, donde se determina qué tuplas nuevas deben ser insertadas, borradas, o modificadas en la tabla materializada.
- La tabla materializada se mantiene por el tiempo en que se consulta, si la vista no es consultada por un periodo de tiempo, entonces es borrada.

Operaciones DML sobre Vistas

- Las operaciones sobre vistas deben convertirse en operaciones equivalentes sobre las tablas base subyacentes.
- La forma en que se logra esta correspondencia es mediante el álgebra relacional, dado que la definición de vista es una expresión algebraica, con nombre. Este proceso, de sustitución, funciona debido a la propiedad de cierre del álgebra .
- Para unos casos muy simples, las vistas se pueden actualizar. Además el comando "instead of" de triggers puede ser usado para hacer que modificaciones en una vista se traduzcan en modificaciones en una tabla.

Borrar vistas

- El borrar una vista con DROP viewname; borra la definición de una vista.
- El borrar una vista no altera de ninguna forma las tuplas que la definen.
- Sin embargo, si se borra la relación subyacente a la view, hace que las vista que define queden inútiles.

Actualización de Vistas: INSERT | DELETE | UPDATE

- No todas las vistas se pueden actualizar. Lo anterior, va a depender del tipo de vista. Así, tenemos los siguientes tipos: (1) Vistas de subconjunto de columnas, (2) Vistas de subconjunto de filas, (3) Vistas de reunión, (4) Vistas de resumen estadístico

Vistas de subconjunto de columnas (1/2)

Si la vista ha sido creada incluyendo la clave primaria de la relación subyacente, es actualizable. Por el contrario, si no la incluye, no es actualizable. Considere la siguiente table de Proveedores, donde ID_PROV es el id, NOMBRE el nombre, SITUACION es un código de estado y CIUDAD la ciudad de proveedores

ID_PROV	NOMBRE	SITUACION	CIUDAD
S1	LAPIZ LOPEZ	20	CONCEPCION
S2	TORRE	10	SANTIAGO
S3	REHIN	30	SANTIAGO
S4	PELIKAN	20	CONCEPCION
S5	XEROX	30	TEMUCO

- VISTA 1:
  CREATE VIEW ID_CIUDAD
  AS SELECT ID_PROV,CIUDAD **FROM** PROV;
- VISTA 2:
  CREATE VIEW SITUACION_CIUDAD
  AS SELECT SITUACION,CIUDAD **FROM** PROV;

Vistas de subconjunto de columnas 2/2

- VISTA 1: **CREATE VIEW** ID_CIUDAD
  AS SELECT ID_PROV,CIUDAD **FROM** PROV;
- Se puede insertar un nuevo proveedor en la vista, tal como (S6, OSORNO), insertando el registro (S6, NULL,NULL,OSORNO) en la tabla subyacente PROV.
- Se puede eliminar un registro existente de la vista, tal como (S1, CONCEPCION), eliminando el registro (S1, LAPIZ LOPEZ, 20, CONCEPCION) de la tabla subyacente PROV.
- Se puede modificar un registro de la vista, tal como, cambiar la ciudad de un proveedor, modificándose el registro correspondiente de la tabla subyacente PROV.
- VISTA 2: **CREATE VIEW** SITUACION_CIUDAD
  AS SELECT SITUACION,CIUDAD **FROM** PROV;
- Insertar un nuevo registro en la vista, tal como (40, VALPARAISO), el sistema intentará insertar el registro (NULL, NULL, 40, VALPARAISO) en PROV. Esto no será posible ya que la clave primaria no puede ser NULL.
- Al eliminar un registro de la vista, no se sabrá cuál es el registro que corresponde a la tabla PROV, ya que no se ha especificado el identificador de proveedor (ID_PROV). Este no forma parte de la vista.
- Sucede lo mismo que en 2, al tratar de modificar un registro de la vista.

Vistas de subconjunto de filas

En estos casos sucede lo mismo que las vistas de subconjunto de columnas, de tal manera que si incluyen la clave primaria, será posible actualizarlas

ID_PROV	NOMBRE	SITUACION	CIUDAD
S1	LAPIZ LOPEZ	20	CONCEPCION
S2	TORRE	10	SANTIAGO
S3	REHIN	30	SANTIAGO
S4	PELIKAN	20	CONCEPCION
S5	XEROX	30	TEMUCO

- VISTA 1:
```
CREATE VIEW PROV_CONCEPCION
AS SELECT SITUACION,CIUDAD FROM PROV
WHERE CIUDAD= 'CONCEPCION';
```

- Esta vista si no es actualizable ya que el ID_PROV no es

Vistas de Join

$$
\begin{aligned}
S &= (\underline{S1}, S2, S3, S4) \\
P &= (\underline{P1}, P2, P3, P4) \\
S \bowtie P &= (\underline{S1}, S2, S3, S4, \underline{P1}, P2, P3, P4)
\end{aligned}
$$

Vistas de resumen estadístico

CREATE VIEW PC (P, CANTTOTAL)
AS SELECT P, SUM(CANT) **FROM** SP
GROUP Y P;

Esta vista no permite modificaciones ni inserciones en el atributo CANTTOTAL

Resumen: Vistas actualizables

En general, SQL permite modificación sobre vistas cuando vistas son definidas por selecćon (SELECT y no SELECT DISTINCT) de algunos atributos de una relación R (la cual puede a su vez ser una vista actualizable) con las siguientes condiciones:

- La cláusula WHERE no debe envolver una subconsulta en R
- La cláusura FROM debe contener una ocurrencia de R
- La lista en SELECT debe incluir suficientes atributos que por cada tupla insertada en al vista, se puede llenar los otros atributos con valores NULOS o el defecto definido (debe poderse identificar únicamente la tupla a insertar).
- Los únicos valores insertados son los que deben aparecer en el SELECT.
- Una inserción puede no tener resultados sobre la vista si es que la tupla insertada no cumple la condición de la definición de la vista.

Resumen: Vistas actualizables (cont.)

- Se pueden también borrar vistas actualizables. El borrar, al igual que insertar, se pasa a la relación R subyacente. Sin embargo, para aseguar que solo las tuplas que son vistas sean las que se borran, se agrega (usando AND) la condición del WHERE de la vista en el WHERE del borrado.
- Similarmente para el update, se actualizan las tuplas que cumplen la condición del view.

Resumen Vistas:Ventajas

- Ofrecen independencia lógica de datos, en casos de reestructuración de la base de datos. La independencia lógica ha de entenderse como la independencia de los usuarios con respecto a la estructura lógica de la base de datos.
- Permiten a diferentes usuarios ver los mismos datos de distintas maneras y al mismo tiempo.
- Se simplifica la percepción del usuario, ya que estos sólo se concentran en aquellos datos que les son interesantes.
- Son un mecanismo de seguridad, debido a que habrán datos ocultos (los no visibles a través de la vista). Dichos datos están a salvo de accesos.

Instead-of de Triggers sobre Vistas

- Cuando un trigger es definido sobre una vista, se puede usar INSTEAD OF en lugar de BEFORE o AFTER para que cuando un evento active un trigger, la acción del trigger se realice en vez del event en sí mismo. Es decir, el instead-of intercepta un intento de modificación de una vista y en su lugar, realiza cualquier acción el diseñador de la base de datos intenta hacer.

Instead-of de Triggers sobre Vistas (cont.)

```
CREATE VIEW EmpleadosMujeres AS
  SELECT nombre, apellido, salario
  FROM EMPLEADO
  WHERE sexo='F';
```

Este ejemplo hace que cuando se insertar tuplas, el sistema no puede deducir que el empleado es en efecto una mujer ya que sexo no es parte del la cláusua SELECT. Entonces se puede usar INSTEAD-OF para suplir este efecto.

Instead-of de Triggers sobre Vistas (cont.)

```
CREATE TRIGGER EmpleadosMujerInsert AS
INSTEAD OF INSERT ON EmpleadosMujer
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
INSERT INTO EMPLEADO(nombre,apellido,salario, sexo)
VALUES(NewRow.nombre, NewRow.apellido, 'F', NewRow.salario);
```

Vista Materializadas: costo de mantención

```
CREATE MATERIALIZED VIEW EmpleadosMujeres AS
  SELECT nombre, apellido, salario
  FROM EMPLEADO
  WHERE sexo='F';
```

En principio un DBMS necesita recalcular una vista materializada cada vez que una de sus tablas bases en actualizada. Para vistas simples, es posible limitar el número de veces que uno necesita recalcular la vista.

Vista Materializadas: costo de mantención (cont.)

- No se necesita actualizar la vista cuando se actualizan atributos que no están en la definición de la vista.
- La idea básica es que todas las actualizaciones sobre las vistas materializadas sean incrementales. Es decir, no se debe reconstruir la vista completa.
- Para delete, insert y udpate, estas actualizaciones se pueden traducir en una o más consultas sobre las vistas. Más aún, estas actualizaciones solo afectan algunas de las tuplas en las vistas.

Vista Materializadas: mantención periódica

- Cuando la actualización de una base es muy frecuente, manejar vista puede ser mus costoso.
- En esos casos, se puede considerar actualizaciones periódicas de manera de manejar esperas entre vistas actualizadas pero que no afectan significativamente un proceso o análisis de datos.

Vista Materializadas: Reescritura de consultas

Existe una simple regla para saber cuándo una consulta puede ser reescrita para usar materializadas vistas. Considere:

Vista	**Consulta**
SELECT L_V	SELECT L_Q
FROM R_V	FROM R_Q
WHERE C_V	WHERE C_Q

Vista Materializadas: Reescritura de consultas (cont.)

Vista	**Consulta**
SELECT L_V	SELECT L_Q
FROM R_V	FROM R_Q
WHERE C_V	WHERE C_Q

- Las relaciones en R_V aparecen todas en R_Q.
- La condición C_Q es equivalente a C_V [AND C], para algún C.
- Si C es necesario, entonces los atributos de la relación en R_V que C menciona son atributos en la lista L_V
- Atributos en L_Q que vienen de las relaciones en la lista R_V están también en la lista L_V.

Entonces se puede reescribe Q para usar V como sigue:

- Reemplace la lista R_Q por vista V y las relaciones que están en la lista R_Q pero no en R_V.
- Reemplace C_Q por C. Si C no es necesario ($C_V = C_Q$), entonces no existe el WHERE.

Vista Materializadas: Reescritura de consultas - Ejemplo

Movie(title,year, length, genre,studioName,producer)
StarIn(titleMovie,starName, movieYear)
MovieExec(name,address, netWorth)

Se define una vista materializada y consulta de la siguiente forma:

Vista

```
SELECT title, year,name
FROM Movie,MovieExec
WHERE producer = name
```

Consulta

```
SELECT startName
FROM StarIn, Movie,MovieExec
movieTitle = title AND
producer = name AND
name = 'Bialystock' AND
movieYear = year;
```

Vista Materializadas: Reescritura de consultas - Ejemplo (cont.)

Vista

```
SELECT title, year,name
FROM Movie,MovieExec
WHERE producer = name
```

Consulta

```
SELECT startName
FROM StarIn, Movie,MovieExec
movieTitle = title AND
producer = name AND
name = 'Bialystock' AND
movieYear = year;
```

Reescritura:

```
SELECT startName
FROM StarIn, MovieProd
WHERE movieTitle = title AND movieYear = year AND
  name = 'Bialystock';
```

Vista Materializadas: creación automática

Uno puede pensar es usar una creación automática de vistas materializadas de manera de agilizar el proceso de consultas. En principio cualquier consulta puede ser considerada una vista, pero eso no tiene sentido. Se puede restringir este número considerando que se crean vista que:

- Tienen una lista de relaciones en el FROM que son un subconjunto de aquellas en el FROM de al menos una consulta de la carga de trabajo establecida.
- Tienen un WHERE que es un AND de condiciones que cada una aparece al menos en una consulta.
- Tiene una lista de atributos en el SELECT que son suficientes para ser usados en al menos una consulta.

Nota: Consideraciones adicionales para decidir si una vista materializada es conveniente es el tamaño de relaciones que produce.

Indices: Motivación

- Un índice sobre un atributo de una relación es una estructura de datos que hace eficiente encontrar las tuplas que tienen un valor del atributo.
- Cuando la relación es muy grande, entonces se hace costoso tener que recorrer toda la relación.
- Uno de los índices más comunes es el Btree.

Indices: Declaración

Aunque los índices no son parte del estándar de SQL hasta SQL99, la mayoría de los motores de bases de datos permiten crear índices sobre ciertos atributos. Esto se puede hacer, por ejemplo, sobre el atributo year de la relación Movie de la siguiente forma:

```
CREATE INDEX YearIndex On Movie(year);
```

Muchos de los motores también permiten definir índices sobre múltiples atributos (ej. claves de múltiples atributos). En esos casos uno puede pensar en un order particular para los atributos combinados que sea más conveniente, dejando como primer atributo el que sea más utilizado en forma individual.

Indices: Selección

Dos factores importantes son:

- La existencia de un índice sobre un atributo puede acelerar la ejecución de consultas en la cual un valor o rango de valores es especificado para este atributo, y puede acelerar operaciones de join que involucren este atributo también.
- Por otro lado, cada operación de actualización en una relación que contiene algún índice sobre sus atributos se hace más compleja y costosa.

Indices: Selección (cont.)

- Para entender como seleccionar un índice, es necesario conocer el costo de contestar una consulta. Por el momento, sepamos que las tuplas se almacenan generalmente distribuidas en páginas en el disco, donde una página generalmente almacena varios bytes, y consecuentemente, posiblemente varias tuplas.
- Para examinar cada tupla, se requiere traer toda la página en memoria principal. Por otro, cuesta un poquito más examinar toda las tuplas de una página que solo una tupla.
- En el extremo, se requiere revisar todas las páginas en disco.

Indices: Uso

Lo más frecuente es hacer índices sobre las claves. Esto porque:

- Consultas en las cuales el valor de la clave es especificado es común.
- Debido a que a lo más hay una tupla por un valor de la clave, los índices retornan una o ninguna tupla. Entonces, a lo más se requiere recuperar una página a memoria principal.

Además, hay dos casos en los cuales un índice para atributos no claves puede set recomendable:

- Si el atributo es casi una clave (pocas tuplas tienen el mismo valor) no se necesita recuperar muchas páginas.
- Si las tuplas están agrupadas por ese valor.

Indices: cálculo de costo

- Para determinar el costo de crear un índice debemos hacer ciertas presunciones acerca de cuales consultas y modificaciones son las más probables de ocurrir en la base de datos.
- Ejemplo de datos a considerar son: número de páginas por relación, algún tipo de información de la distribución de valores del atributo de interes en las tuplas, modificaciones necesita leer y escribir páginas.

Indices: cálculo de costo (cont.)

Suponga que hay tres operaciones frecuentes en la base de datos sobre la relación StarIn:

- Consulta 1 (Q1):

  SELECT movieTitle, movieYear FROM StarIn WHERE starName = s;

- Consulta 2 (Q2):

  SELECT starName FROM StarIn WHERE movieTitle = t AND movieYear = y;

- Insertar (I):

  INSERT INTO StarIn VALUES(t,y,s);

Indices: cálculo de costo (cont.)

Las presunciones o información de la base de datos son

- StarIn tiene 10 páginas.
- En promedio, una estrella aparece 3 veces en una película y una película tiene 3 estrellas.
- Debido a que las tuplas de una estrella en particular se distribuyen sobre las 10 páginas, aunque se tenga un índice sobre starName, entonces, en promedio se necesitan 3 lecturas para encontrar en promedio 3 tuplas de estrellas o películas.
- Actualizaciones requieren una lectura y escritura de página.
- Se asume que, incluso sin índices, se puede encontrar una página donde poder insertar una tupla, sin recorrer toda la relación.

Indices: cálculo de costo (cont.)

Acción	No Indice	Indice sobre estrella	Indice sobre película	Ambos
Q1	10	4	10	4
Q2	10	10	4	4
I	2	4	4	6

- Se recorre toda tabla cuando no hay índices por el que buscar
- Actualización requiere una lectura y una escritura
- Además de la lectura de la tabla se debe leer el índice (o actualizarlo en caso necesario). Eso hace que el costo sea 4 y no 3 para las consultas con índices.

Recursividad: Introducción

El SQL-99 estándar incluye reglas recursivas, tal como lo hace Datalog (bases de datos deductivas). Sin embargo, la forma de cómo la recursividad en SQL es introducida dependen del sistema de administrador de base de datos.

Preliminares: Reglas recursivas

Ancestro

Persona	Ancestro
X	Z
Y	Z
Z	Q
Q	R

Se pude definir una bases de datos en forma intensional con las siguientes reglas:

$$Ancestros(x, y) \leftarrow Ancesto(x, y)$$
$$Ancestros(x, y) \leftarrow Ancestro(x, z) \text{ AND } Ancestros(z, y)$$

Preliminares: Reglas recursivas (cont.)

Diferentes formas de recursividad:

- Forma recursiva derecha:

$$Ancestros(x, y) \leftarrow Ancesto(x, y)$$
$$Ancestros(x, y) \leftarrow Ancestro(x, z) \text{ AND } Ancestros(z, y)$$

- Forma recursiva izquierda:

$$Ancestros(x, y) \leftarrow Ancesto(x, y)$$
$$Ancestros(x, y) \leftarrow Ancestros(x, z) \text{ AND } Ancestro(z, y)$$

- Forma recursiva no lineal:

$$Ancestros(x, y) \leftarrow Ancesto(x, y)$$
$$Ancestros(x, y) \leftarrow Ancestros(x, z) \text{ AND } Ancestros(z, y)$$

Preliminares: Negación en reglas recursivas

A veces es necesario manejar negación en una recusividad. Hay formas seguras e inseguras de manejar negación en la recursividad. En el siguiente ejemplo, considere una relación $Vuelo(x, y, z, d, r)$, donde x es aerolínea, y desde, z hacia, d hora de partida y r hora de llegada. Para este esquema una regla recusiva de Conecta es:

$$Conecta(x, y) \leftarrow Vuelo(a, x, y, d, r)$$
$$Conecta(x, y) \leftarrow Conecta(x, z) \text{ AND } Conecta(z, y)$$

Preliminares: Negación en reglas recursivas (cont.)

Considerando solo vuelos en LAN:

$$LanConecta(x, y) \leftarrow Vuelo(LAN, x, y, d, r)$$
$$LanConecta(x, y) \leftarrow LanConecta(x, z) \text{ AND } LanConect(z, y)$$

Considerando solo vuelos en Sky:

$$SkyConecta(x, y) \leftarrow Vuelo(Sky, x, y, d, r)$$
$$SkyConecta(x, y) \leftarrow SkyConecta(x, z) \text{ AND } SkyConecta(z, y)$$

Luego, uno podría querer saber las ciudad que conecta LAN pero no SKY

$$SoloLan(x, y) \leftarrow LanConecta(x, y) \text{ AND NOT} SkyConecta(x, y)$$

En este caso, la regla funciona bien.

Preliminares: Negación en reglas recursivas (cont.)

Sin embargo, la siguiente regla ya no funciona:

$$P(x) \leftarrow R(x) \text{AND NOT } Q(x)$$
$$Q(x) \leftarrow R(x) \text{AND NOT } P(x)$$

Preliminares: Negación en reglas recursivas (cont.)

En general, se restringe la negación en recusividad cuando la negación es estratificada (stratified). Esta propiedad se define com:

- Dibuje un grafo cuyos nodos correspondan a los predicados o relaciones de la base de datos intensional
- Dibujo un arco desde A a B si una regla con predicado A en el consecuente tiene una negación en el antecedente con predicado B. Coloque el signo $-$ a ese arco.
- Dibujo un arco desde A a B si una regla con predicado A en el consecuente no tiene una negación en el antecedente con predicado B.
- Si el grafo tiene un ciclo conteniendo uno más signos negativos, entonces la negación no es estratificada.

Definiendo bases de datos intensionales en SQL: WITH

La sentencia WITH permite la definición de relaciones intensionales (las que son derivadas al aplicar uno o más reglas). Una forma simple de esta sentencia es:

WITH [RECURSIVE] R AS $<$ definición de $R>$ $<$ consulta con $R>$

Además, se pueden definir varias relaciones temporales

WITH
[RECURSIVE] R_1 AS $<$ definicion $R_1>$,
[RECURSIVE] R_2 AS $<$ definicion $R_2>$,
....
[RECURSIVE] R_n AS $<$ definicion $R_n>$
$<$ query involving $R_1, R_2 \ldots R_n>$

Definiendo bases de datos intensionales en SQL: WITH (cont.)

```
(1)  WITH RECURSIVE Conecta(desde, hacia) AS
(2)          (SELECT desde, hacia FROM Vuelo)
(3)       UNION
(4)          (SELECT R1.desde, R2.hacia
(5)          FROM Conecta AS R1, Conecta As R2
(6)          WHERE R1.hacia = R2.desde)
(7)  SELECT * FROM Conecta;
```

Negación estratificada

```
(1)     WITH
(2)         Triples As SELECT linea, desde, hacia FROM vuelo,

(3)         RECURSIVE Conecta(linea, desde, hacia) AS
(4)             (SELECT * FROM Triples)
(5)         UNION
(6)             (SELECT Triples.linea, Triples.desde, Triples.hacia
(7)             FROM Triples, Conecta
(8)             WHERE Triples.hacia = Conecta.desde AND
(9)                 Triples.linea = Conecta.linea

(10)        (SELECT desde,hacia FROM Conecta WHERE linea ='Lan')
(11)    EXCEPT
(12)        (SELECT desde,hacia FROM Conecta WHERE linea ='Sky');
```

Negación no estratificada (no permitida)

```
(1)     WITH
(2)         RECURSIVE P(x) AS
(3)             (SELECT * FROM R)
(4)          EXCEPT
(5)             (SELECT * FROM Q)

(6)         RECURSIVE Q(x) AS
(7)             (SELECT * FROM R)
(8)          EXCEPT
(9)             (SELECT * FROM P)

(10)    SELECT * FROM P;
```