# Bases de Datos

**Gabriela Arévalo**

garevalo@unq.edu.ar

# SQL: Structured Query Language

- SQL es el lenguaje comercial más aceptado
- Standard aprobado por ANSI e ISO en 1986
- Es un lenguaje no procedural cuyo objetivo es el de soportar la:
  - Definición
  - Manipulación
  - Control de Datos
- Su poder expresivo incluye al AR y lo extiende

# Características

- Opera sobre “conjuntos de tuplas”
  - Incluso para las operaciones de inserción, borrado y actualización
- No elimina automáticamente tuplas repetidas
- El mismo lenguaje puede ser usado en forma:
  - Interactiva
  - Embebida en un lenguaje de programación

# Sublenguajes

- DDL (Data Definition Language)
  - CREATE
  - ALTER
  - DROP
- DML (Data Manipulation Language)
  - INSERT
  - UPDATE
  - DELETE
  - SELECT => es el único comando que permite recuperar tuplas almacenadas en la base

# Mecanismos

- Seguridad
  - A veces se lo llama DCL (Data Control Language)
  - Grant (Permisos sobre objetos y logins)
- Concurrencia y Recuperación
  - Commit work
  - Rollback work
- Diccionario de Datos
  - Organizado como conjunto de tablas pudiendo ser accedido con la misma sintaxis, por los usuarios que tienen permiso

# DDL: Data Definition Language

- La instrucción CREATE TABLE:
  - Crea una tabla vacía
  - Genera información en el Diccionario de Datos
  - Algunos elementos de la instrucción CREATE TABLE:
    - Nombre de la tabla
    - Nombre de cada atributo
    - Tipo de datos para cada atributo
    - Restricciones de integridad: not null, primary key, foreign key

# DDL

- ✓ Comencemos con el ejemplo de las facultades y estudiantes que pertenecen a clubes.
- ✓ Cómo definimos el esquema de nuestra BD usando SQL?

# Ejemplo: Definición de Tablas

```
CREATE TABLE department
(   code    varchar(3)   PRIMARY KEY,
    name    varchar(40)   NOT NULL,
    chair   varchar(11) );

CREATE TABLE person
(   pId      varchar(11) PRIMARY KEY,
    dob      date NOT NULL,
    firstName varchar(20) NOT NULL,
    lastName  varchar(20) NOT NULL);

CREATE TABLE faculty
(   pId      varchar(11)PRIMARY KEY,
    rank     varchar(10) NOT NULL,
    dept     varchar(3) NOT NULL,
    CONSTRAINT rankValue CHECK (rank IN ('Assistant', 'Associate', 'Full',
    'Emeritus')),
    CONSTRAINT facultyPIdFk FOREIGN KEY (pId) REFERENCES person (pId),
    CONSTRAINT facultyDeptFk FOREIGN KEY (dept) REFERENCES department
    (code));

ALTER TABLE department
ADD CONSTRAINT departmentChairFk FOREIGN KEY(chair)
    REFERENCES faculty(pId) ON DELETE SET NULL;
```

# Ejemplo: Definición de Tablas

```
CREATE TABLE student
(  pId      varchar(11)   PRIMARY KEY,
   status varchar(10)   NOT NULL,
   major  varchar(3)     NOT NULL,
   CONSTRAINT statusValue CHECK (status IN ('Freshman', 'Sophomore',
   'Junior', 'Senior',  'Graduate')),
   CONSTRAINT studentPIdFk FOREIGN KEY (pId) REFERENCES person(pId),
   CONSTRAINT studentMajorFk FOREIGN KEY (major) REFERENCES department
   (code));
CREATE TABLE campusclub
(  cId    varchar(10)   PRIMARY KEY,
   name   varchar(50)   NOT NULL,
   phone     varchar(12),
   location  varchar(40),
   advisor   varchar(11),
CONSTRAINT campusclubAdvisor_Fk FOREIGN KEY(advisor) REFERENCES faculty
   (pId) ON DELETE SET NULL);
CREATE TABLE clubs
(  pId  varchar(11)  NOT NULL,
   cId  varchar(10)  NOT NULL,
   CONSTRAINT clubsPk PRIMARY KEY(pId, cId),
   CONSTRAINT clubsPIdFk FOREIGN KEY (pId) REFERENCES student(pId) ON
   DELETE CASCADE,
   CONSTRAINT clubsCIdFk FOREIGN KEY (cId) REFERENCES campusclubs(cId)
   ON DELETE CASCADE);
```

# Tipos de Datos

```
CREATE TABLE person
( pId        varchar(11) PRIMARY KEY,
  dob  date NOT NULL,
   firstName varchar(20) NOT NULL,
   lastName  varchar(20) NOT NULL);
```

# Tipos de Datos

| Category | Data types |
|---|---|
| Character String | char, varchar, nchar, nchar varying, clob (character large object), nclob |
| Number | int,integer, smallint, numeric, decimal, float, real, double precision |
| Temporal | date, time, timestamp, interval |
| Boolean | boolean (true/false/unknown) |
| Constructed | User-defined types (UDTs) |
| | <reference type>: ref udt_name |
| | <row type>: row (one or more (Field, <data_type>) pairs) |
| | <collection type>: array[] |
| Binary String | blob (binary large object) |
| Bit String | bit, bit varying |

# Restricciones de Integridad

```
CREATE TABLLE person
( pId          varchar(11) PRIMARY KEY,
  dob          date NOT NULL,
  firstName    varchar(20) NOT NULL,
  lastName     varchar(20) NOT NULL);

CREATE TABLE clubs
( pId       varchar(11)   NOT NULL,
  cId       varchar(10)   NOT NULL,
  CONSTRAINT clubsPk PRIMARY KEY(pId, cId),
  ….);
```

# Restricciones de Integridad

- ✓ **NOT NULL**: especifica que el valor de la columna nunca puede ser nulo.
- ✓ **PRIMARY KEY**: especifica el atributo (o conjunto de atributos) que definen la clave primaria en una tabla.
- ✓ **UNIQUE**: define que el valor de un atributo (o conjunto de atributos) que es único, es decir dos tuplas no tienen el mismo valor en la tabla.

# Restricciones de Integridad

```
CREATE TABLE faculty
( pId          varchar(11) PRIMARY KEY,
  rank    varchar(10) NOT NULL,
  dept    varchar(3) NOT NULL,
  CONSTRAINT rankValue CHECK (rank IN
  ('Assistant', 'Associate', 'Full',
  'Emeritus')),
  …);
```

# Restricciones de Integridad

- ✓ CHECK: define restricciones conocidas como restricciones de dominio sobre el valor de la columna.

# Restricciones de Integridad

```
CREATE TABLE faculty
( pId    varchar(11) PRIMARY KEY,
  rank   varchar(10) NOT NULL,
  dept   varchar(3) NOT NULL,
  CONSTRAINT rankValue CHECK (rank IN
  ('Assistant', 'Associate', 'Full',
  'Emeritus')),
  CONSTRAINT facultyPIdFk FOREIGN KEY (pId)
  REFERENCES person (pId),
  CONSTRAINT facultyDeptFk FOREIGN KEY
  (dept) REFERENCES department (code));
```

# Restricciones de Integridad

- ✓ FOREIGN KEY: define la noción de integridad referencial entre una clave foránea y una clave primaria en una tabla separada. Ambas claves deben tener el mismo tipo de datos.

# DDL

- Otras instrucciones sobre tablas:
  - ALTER TABLE: permite modificar la definición de la tabla
    - cambiar nombre de atributos
    - cambiar tipos de atributos
    - agregar o quitar atributos
    - agregar o quitar restricciones de integridad
    - etc ...
  - DROP TABLE: permite eliminar una tabla

# Ejemplo: ALTER

```
ALTER TABLE student
   ADD gradePointAverage numeric
   ALTER status SET DEFAULT =
     'Freshman';
```

**‘ Agrega el nuevo atributo (o columna) *gradePointAverage* a la tabla student y modifica el valor por defecto del atributo *status* a Freshman’**

# Ejemplo: ALTER

```
ALTER TABLE student
   ADD gradePointAverage numeric
   ALTER status SET DEFAULT =
     'Freshman';
```

**‘ Agrega el nuevo atributo (o columna) *gradePointAverage* a la tabla student y modifica el valor por defecto del atributo *status* a Freshman’**

# Ejemplo: ALTER

```
CREATE TABLE department
(        code       varchar(3)   PRIMARY KEY,
   name     varchar(40)   NOT NULL,
   chair    varchar(11) );

CREATE TABLE person
(        pId         varchar(11) PRIMARY KEY,
   dob       date NOT NULL,
   firstName varchar(20) NOT NULL,
   lastName  varchar(20) NOT NULL);

CREATE TABLE faculty
(        pId         varchar(11)PRIMARY KEY,
   rank      varchar(10) NOT NULL,
   dept      varchar(3) NOT NULL,
   CONSTRAINT rankValue CHECK (rank IN ('Assistant', 'Associate', 'Full',
'Emeritus')),
   CONSTRAINT facultyPIdFk FOREIGN KEY (pId) REFERENCES person (pId),
   CONSTRAINT facultyDeptFk FOREIGN KEY (dept) REFERENCES department
(code));

ALTER TABLE department
ADD CONSTRAINT departmentChairFk FOREIGN KEY(chair)
REFERENCES faculty(pId) ON DELETE SET NULL;
```

# Ejemplo: DROP

```
DROP TABLE student RESTRICT;
DROP TABLE student CASCADE;
```

- ✓ **RESTRICT:** cuando una columna se elimina, esta opción se usa para indicar que la columna se pueda eliminar si no hay otro componente del esquema que se refiera a esa columna.
- ✓ **CASCADE:** con esta opción, se elimina todos los componentes del esquema que hace referencia a la columna eliminada.

# DML: Data Manipulation Language

- Modificación de los datos
  - INSERT: permite agregar tuplas a una tabla
  - UPDATE: permite cambiar tuplas de una tabla
  - DELETE: permite borrar tuplas de una tabla

# Ejemplo: INSERT

```
INSERT INTO department
   VALUES ('cse', 'Computer Science and
   Engineering', '111-11');
```

- ✓ Un valor se especifica para cada columna de la tabla en el orden de las columnas de la tabla. Se puede usar un valor NULL para columnas con valores desconocidos.

```
INSERT INTO department(code, name, chair)
    VALUES ('cse', 'Computer Science and
   Engineering', '111-11');
```

- ✓ Se especifica sólo las columnas a los cuales se les asignará los valores especificados. Las columnas no especificadas toman el valor NULL salvo cuando se haya dado un valor por defecto.

# Ejemplo: DELETE y UPDATE

```
DELETE FROM student
WHERE status = 'Junior';
```

- ✓ Se borran de la tabla todas las tuplas que cumplan con la condición especificada en el *where*. En este caso, que el status del estudiante sea 'Junior'

```
UPDATE faculty
     SET rank = 'full professor'
     WHERE name = 'Joseph';
```

- ✓ Se actualiza todas las tuplas con el rango de 'full professor' a todos los miembros de la facultad con nombre Joseph.

# Ejemplo: Restricciones de Integridad

```
CREATE TABLE campusclub
(  cId   varchar(10)    PRIMARY KEY,
   name         varchar(50)    NOT NULL,
phone      varchar(12),
location   varchar(40),
advisor    varchar(11),
CONSTRAINT campusclubAdvisor_Fk FOREIGN KEY(advisor)
   REFERENCES faculty(pId) ON DELETE SET NULL);

CREATE TABLE clubs
(  pId varchar(11)   NOT NULL,
   cId varchar(10)   NOT NULL,
   CONSTRAINT clubsPk PRIMARY KEY(pId, cId),
   CONSTRAINT clubsPIdFk FOREIGN KEY (pId) REFERENCES
   student(pId) ON DELETE CASCADE,
   CONSTRAINT clubsCIdFk FOREIGN KEY (cId) REFERENCES
   campusclubs(cId) ON DELETE CASCADE);
```

# Restricciones de Integridad

- La restricción de la foreign key incluye las acciones que deben realizarse para asegurarse la preservación de la integridad referencial cuando se actualizan o eliminan tuplas.
- Las acciones son:
  - NO ACTION
  - CASCADE
  - SET NULL
  - SET DEFAULT

# Restricciones de Integridad

- ✓ **NO ACTION**: Es por defecto. Se evitará un update or delete si la integridad referencial se viola.
- ✓ **CASCADE:** Un update o delete de una tupla en la tabla referenciada hará que la base de datos actualice o borre todas las filas que “matchean” en la tabla que referencia.

# Restricciones de Integridad

- ✓ **SET NULL:** Un update o delete de una tupla en la tabla referenciada hará que la base de datos anote un valor nulo sobre la foreign key en todas las filas que “matchean” en la tabla que referencia. Esta acción no puede usarse con la restricción de not null en la foreign key.
- ✓ **SET DEFAULT:** Un update o delete de una tupla en la tabla referenciada hará que la base de datos anote el valor por defecto sobre la foreign key en todas las filas que “matchean” en la tabla que referencia.

# DML: Recuperación de datos

- SELECT:
  - Devuelve tuplas de la base

```
SELECT a1, ...., an
FROM r1, ..., rm
WHERE p
```

donde:

a1, ..., an son nombres de atributos (puede usarse *)

r1, ..., rm son nombres de tablas

p es una condición

  - Expresa en álgebra relacional:

$\pi_{A1, ..., An} (\sigma_p (R1 \times ... \times Rm))$

# DML: Ejemplo

- Comencemos con un ejemplo
  - Sea la BD:
    - `fabs (#fab, nombre, dirección)`
    - `prods (#prod, descripción)`
    - `ventas (#fab, #prod, precio)`
- Cómo definimos el esquema de nuestra BD usando SQL?

# Ejemplo

```
CREATE TABLE fabs
   (#fab number(5) NOT NULL,
   nombre char(20),
   direccion char(40));

CREATE TABLE prods
(#prod number(5) NOT NULL,
   descripcion char(20));

CREATE TABLE ventas
   (#fab number(5) NOT NULL,
    #prod number(5) NOT NULL,
   precio float);
```

# Ejemplos

- Dar el precio al que vende el producto 7 el fabricante 2

```
SELECT precio
FROM ventas
WHERE #fab = 2 AND #prod = 7
```

- Dar el número de los fabricantes que vendieron el producto 4 a menos de $ 100

```
SELECT #fab
FROM ventas
WHERE precio < 100 AND #prod = 4
```

# DML: Ordenamiento de tuplas

- Dar los #prod de los productos fabricados, pero ordenados en forma ascendente

```
SELECT #prod
FROM ventas
ORDER BY #prod asc;
```

- La cláusula ORDER BY permite indicar el campo por el cual se ordena el resultado.

- Se indica el orden (asc o desc)

# DML: Filtrado de repetidos

- ✓ Dar los #prod de los productos fabricados, filtrando los repetidos

```
SELECT DISTINCT #prod
FROM ventas
ORDER BY #prod
```

- ✓ La cláusula DISTINCT filtra tuplas repetidas

# DML: Join

- Dar los nombres de fabricantes y los #prod que venden

```
SELECT nombre, #prod
FROM ventas, fabs
WHERE ventas.#fab = fabs.#fab;
```

- Dar los nombres de fabricantes y los #prod que venden, tales que vendió el producto a $100.

```
SELECT nombre, #prod
FROM ventas, fabs
WHERE ventas.#fab = fabs.#fab and precio
= 100;
```

# DML: Union

- Dar los #fab que viven en “d1” o que venden algún producto a precio mayor que $ 500.

```
SELECT #fab FROM fabs
WHERE direc = “d1”
UNION
SELECT #fab FROM ventas
WHERE precio > 500;
```

- La unión elimina tuplas repetidas

# DML: Resta

- ✓ Dar los #fab que no venden ningún producto

```
SELECT #fab FROM fabs
MINUS
SELECT #fab FROM ventas;
```

O también:

```
SELECT #fab FROM fabs
WHERE #fab NOT IN
   (SELECT #fab
   FROM ventas)
```

# DML: Funciones de Suma

- Las consultas anteriores se podían resolver en álgebra relacional
- El SQL incluye además funciones para contar tuplas, sumar cantidades, etc.
- Ejemplo:
  - Decir cuántos fabricantes hay

```
SELECT count(#fab)
FROM fabs;
```

  - Dar el precio máximo de venta

```
SELECT max(precio)
FROM ventas;
```

# DML: Operadores aritméticos

- Para cada producto dar su número y su peso en gramos (supongamos que el peso está en kg.)

```
SELECT #prod, peso * 1.000
FROM prods
```

- Observaciones:
  - Expresiones en la cláusula select.
    - Se pueden utilizar expresiones aritméticas en lugar de nombres de atributos

# DML: Operadores aritméticos

- Queremos que el resultado de peso * 1.000 sea redondeado a 2 cifras decimales.

```
SELECT #prod, round(peso * 1000, 2)
FROM prods;
```

  - Existe un conjunto de funciones para el tipo de datos number como por ej. abs(n), mod(m,n), sign(n), etc.
  - Observemos que estas funciones toman el valor de UNA tupla por vez y devuelven un valor para cada una de ellas

# DML: Operadores aritméticos

- Queremos poner un título en la columna:

```
SELECT #prod, round(peso * 1000,2) as
 "Peso en gr."
FROM prods
```

- Se conoce que el precio de la manteca es $ 25 el gramo, queremos listar los productos de nombre manteca y su costo total.

```
SELECT #prod, round(peso * 1000,2) * 25
 as "Precio"
FROM prods
WHERE nombre = "manteca";
```

# DML: Operadores aritméticos

- ✓ Dar los nombres de los productos que pesan 3 veces más que su volumen.

```
SELECT nombre
FROM prods
WHERE peso = 3 * volumen
```

- ✓ Podemos usar expresiones aritméticas como condiciones en el WHERE

# DML: Consultas Anidadas

- Consultas completas que se ubican dentro de la cláusula WHERE de otra consulta
- Ejemplo:

```
SELECT a1, ...,an
FROM r1, ...,rm
WHERE (aj, ...,ak <op-comp>)
      (SELECT bj,...,bk
      FROM s1, ...,sn
      WHERE q)
```

<op-comp> puede ser IN, = ANY, > ANY, = ALL, > ALL.

# DML: Consultas anidadas

- ✓ Dar los nombres de los fabricantes que venden productos que además son vendidos por el fabricante número 1.

```
SELECT f.nombre
FROM fabs f, ventas v
WHERE f.#f = v.#f AND
   f.#f <> 1 AND
   v.#p IN
      (SELECT #p)
      FROM ventas
      WHERE ventas.#f = 1
```

# DML: Función EXISTS

- Chequea si el resultado de una consulta anidada es vacío
- Ejemplo:
  - Dar los nombres de los fabricantes que sólo venden el producto número 2

```
SELECT nombre
FROM fabs
WHERE NOT EXISTS
   (SELECT *
   FROM ventas
   WHERE fabs.#f = ventas.#f
            AND ventas.#p <> 2)
```

# DML: Agrupamiento de tuplas

- Se quiere resolver: Dar, para cada fabricante, la cantidad de productos que vendió
- Esto implica:
  - Tomar cada grupo de tuplas en Ventas correspondiente a un fabricante
  - Contar la cantidad de tuplas en el grupo
- Para realizar esta operación es necesaria la cláusula GROUP BY

# DML: Agrupamiento de tuplas

- Es una cláusula que se agrega al SELECT-FROM-WHERE
- Por ejemplo:

```
SELECT #f, count(*)
FROM ventas
GROUP BY #f
```

- Cómo funciona ?
  - Genera un grupo por cada #fabricante distinto
  - De cada grupo devuelve el #fabricante y la cantidad de tuplas de dicho grupo

# DML: Agrupamiento de Tuplas

- Dar el número de fabricante y los promedios de precios a los cuales vendió

```
SELECT #f, avg(precio)
FROM ventas
GROUP BY #f
```

- Dar los nombres de fabricante y sus totales de precio vendidos

```
SELECT #f, nombre, sum(precio)
FROM fabs, ventas
WHERE fabs.#f = ventas.#f
GROUP BY #f, nombre
```

# DML: Agrupamiento de Tuplas

- Regla sobre el GROUP BY
  - En una sentencia SQL que tiene cláusula GROUP BY, las expresiones en el SELECT pueden ser sólo:
  - Atributos presentes en la cláusula GROUP BY
  - Funciones de agregación sobre atributos
  - Expresiones aritméticas que utilicen los anteriores
- El agrupamiento se realiza después de aplicar el WHERE. O sea sobre las tuplas que cumplen la condición.

# DML: Agrupamiento de tuplas

- Para cada pareja (#f, #p), dar la cantidad de ventas realizadas y el total de $.

```
SELECT #f, #p, count(*), sum
  (precio)
FROM ventas
GROUP BY #f, #p
```

- Idem anterior, pero con precios en U$S (3.00).

```
SELECT #f, nombre, count(*), sum
  (precio)/3.00
WHERE fabs.#f = ventas.#f
GROUP BY #f, nombre
```

# DML: Condiciones sobre grupos

- Con la cláusula HAVING se pueden especificar condiciones sobre los grupos.
- Por ejemplo:
  - Dar el número de fabricante y los promedios de precios a los cuales vendió, pero para los fabricantes con más de 3 ventas.

```
SELECT #f, avg(precio)
FROM ventas
GROUP BY #f
HAVING count(*) > 3;
```

# DML: Condiciones sobre grupos

- ✓ Dar los nombres de fabricantes que hicieron más de 5 ventas sobre productos con #p mayor que 2, junto con el total de precio vendido.

```
SELECT #f, nombre, sum(precio)
FROM fabs, ventas
WHERE fabs.#f = ventas.#f AND #p>2
GROUP BY #f, nombre
HAVING count(*) > 5;
```

# DML: Cambios de formato

- Dar los números de fabricantes y nivel de los fabricantes con nombre Juan. Los queremos ordenados en forma descendiente por nivel.

```
SELECT #fab, nivel
FROM fabs
WHERE nombre = "Juan"
ORDER BY nivel desc
```

- Dar el nombre y dirección de los fabricantes separadas por " - ".

```
SELECT nombre, dirección || ' - ' ||
  ciudad "Domicilio"
FROM FABS
```

- || es el operador de concatenación de string

# DML: Cambios de Formato

- ✓ Mayúsculas y minúsculas
- ✓ Dar los nombres de los productos con descripción = ‘rojos’. No recordamos si la descripción fue ingresado con minúsculas o con mayúsculas.

```
SELECT nombre
FROM prods
WHERE upper(desc) = ‘ROJO’;
```

- ✓ Otra forma equivalente:

```
SELECT nombre
FROM prods
WHERE lower(desc) = ‘ROJO’;
```

# DML: Cambios de formato

- ✓ Funciones sobre STRING
- ✓ Existe un conjunto de funciones que se aplican sobre valores del tipo de datos char(n).
- ✓ Por ej.:

  Subst(s,m[,n]),

  Da el substring de largo n a partir de m tomado de s

  Length(s),

  Da el largo del string s.
- ✓ Son análogas a las funciones definidas sobre number, toman el valor de UNA tupla por vez y devuelven un valor

# DML: valor NULL

- ✓ SQL permite chequear en una consulta si un valor es nulo
- ✓ Predicados: IS NULL, IS NOT NULL
- ✓ Devolver los nombres de los fabricantes de los que no se conoce su dirección

```
SELECT nombre
FROM fabs
WHERE direcc IS NULL
```

# Combinación de Operadores

- ✓ Los operadores update y delete pueden combinarse con select/from/where.
- ✓ Ejemplo: borrar los estudiantes que sean miembros de ClubMed.

```
DELETE FROM student
WHERE pId IN
      (SELECT cl.pId
      FROM clubs cl, campusClub cc
      WHERE cc.name = 'ClubMed'
            AND cc.cId = cl.cId)
```

# Volvemos al DDL

- Vistas
  - Una vista es una tabla virtual que se basa en el resultado de una cláusula SELECT.
  - Sus atributos son atributos de tablas o de otras vistas.
  - Pueden utilizarse una vista en consultas, como si fuera una tabla más.
- Instrucciones sobre vistas:
  - CREATE VIEW
  - ALTER VIEW
  - DROP VIEW

# Ejemplo

- ✓ Volvemos al ejemplo de las Facultades
- ✓ Generamos un grupo de usuarios que necesita ver los nombres de las facultades, rangos y nombres de los departamentos donde trabajan los miembros de la facultad.

```
create view facultyInfo
    select d.name ad dName, p.firstName as
      fName, p.lastName as lName, f.rank as
      rank
    from person p, faculty f, department d
    where f.pId = p.pId and f.dept = d.code
```

# Acciones sobre una Vista

- Sobre una vista puede hacerse consultas

```
select * from facultyInfo
order by dName asc, lName asc,
  fName asc, rank asc;
```

- Las vistas no pueden actualizarse pues pueden crear problemas en el proceso de modificación de la tabla de la base de datos de la cual se generó la vista

# Assertions

- Con check vimos que podemos chequear informacion acerca de una tabla
- `create assertion` nos permite controlar restricciones que implican más de una tabla.

# Assertion: Ejemplo

- Se pide que un miembro de la facultad pueda ser el presidente de solamente el departamento para el cual trabaja.
- Esta restricción implica las tablas faculty y department.

```
create assertion chairDepartment
    check (not exists
        ( select * from faculty, department
          where faculty.pId = department.chair and
          not (faculty.dept = department.code)));
```

- La condición del check debe devolver true o unknown para que se satisfaga la restricción.
- En general, la condición puede ser cualquier expresión condicional y contiene la keyword *exists*

# Assertions

- Las restricciones pueden expresarse como assertions o table constraints.
- El diseñador de la base de datos debe decidir cuál utilizar sabiendo que:
  - Assertions se chequean una vez al final de cada SQL statement.
  - Table constraints se chequean al final de un SQL statement para cada fila en la tabla

# Assertions vs. Table constraints

- La tabla department debe tener siempre al menos una tupla.

```
create table department
    ( ...
      constraint departmentIsNotEmpty
        check ((select count(*) from department) >
      0));
```

- Esta restricción será true siempre, aún cuando la tabla department esté vacía.
- Se debe a que la restricción se checkea solamente si hay filas en la tabla.

# Assertions vs. Table Constraints

- ✓ La tabla department debe tener siempre al menos una tupla.

```
create assertion departmentIsNotEmpty
  check ((select count(*) from department) >
  0);
```

# Assertions

- ✓ Se debe prestar atención cuando la assertion pueda cumplirse cuando la condición evalua a unknown.
- ✓ Ejemplo: se define una restricción donde el promedio de una columna deberia ser mayor a 10

```
create assertion AvgConstraint
        check ((select avg(someColumn) from
          someTable) > 10);
```

- ✓ Si no hay tuplas en la tabla, la función avg devolverá un valor NULL, que retornará una condición con el valor UNKNOWN y se satisface la restricción.