SQL Intermedio 2

Juan F. Pérez

Departamento MACC
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

juanferna.perez@urosario.edu.co

Primer Semestre de 2019

Contenidos

Modificando la base de datos

Borrado de registros (delete)

- Para borrar los registros de una relación que cumplan con un predicado usamos el comando

```
DELETE FROM relacion
WHERE predicado;
```

Borrado de registros (delete)

- Para borrar los registros de una relación que cumplan con un predicado usamos el comando

```
DELETE FROM relacion
WHERE predicado;
```

- Ejemplo

```
DELETE FROM curso
WHERE nombre  = 'Bases de Datos';
```

Borrado de registros (delete)

- Para borrar los registros de una relación que cumplan con un predicado usamos el comando

```
DELETE FROM relacion
WHERE predicado;
```

- Ejemplo

```
DELETE FROM curso
WHERE nombre  = 'Bases de Datos';
```

- Para eliminar todos los registros

```
DELETE FROM curso;
```

Borrado de registros (delete)

- Otro ejemplo de borrado

```
DELETE FROM curso
WHERE creditos BETWEEN 1 AND 3;
```

Borrado de registros (delete)

- Otro ejemplo de borrado

```
DELETE FROM curso
WHERE creditos BETWEEN 1 AND 3;
```

- Ejemplo: borrar todos los cursos con número de créditos por debajo del promedio

```
DELETE FROM curso
WHERE creditos < (
  SELECT avg(creditos)
         FROM curso
         );
```

Insertando registros (insert)

- Insertar un registro en la relación curso (orden de atributos como se definió al crear la tabla)

```
INSERT INTO curso
  VALUES ('000', 'programacion', 'MACC', 4 );
```

Insertando registros (insert)

- Insertar un registro en la relación curso (orden de atributos como se definió al crear la tabla)

```
INSERT INTO curso
  VALUES ('000', 'programacion', 'MACC', 4 );
```

- Se pueden insertar los atributos en otro orden

```
INSERT INTO curso(nombre_unid, nombre, curso_cod,
    creditos)
  VALUES ('MACC','programacion','000', 4 );
```

Insertando registros (insert)

- Insertar un registro en la relación curso (orden de atributos como se definió al crear la tabla)

```
INSERT INTO curso
  VALUES ('000', 'programacion', 'MACC', 4 );
```

- Se pueden insertar los atributos en otro orden

```
INSERT INTO curso(nombre_unid, nombre, curso_cod,
    creditos)
  VALUES ('MACC','programacion','000', 4 );
```

- Se pueden dejar atributos sin especificar (si no violan ninguna restricción)

```
INSERT INTO curso(nombre_unid, nombre, curso_cod,
    creditos)
  VALUES ('MACC','programacion','000', NULL );
```

Insertando registros (insert)

- Se pueden insertar solo algunos atributos (si no violan ninguna restricción)

```
INSERT INTO curso(nombre_unid, nombre, curso_cod)
  VALUES ('MACC','programacion','000');
```

Insertando registros (insert)

- Se pueden insertar solo algunos atributos (si no violan ninguna restricción)

```
INSERT INTO curso(nombre_unid, nombre, curso_cod)
  VALUES ('MACC','programacion','000');
```

- Insertar registros a partir de una consulta

```
INSERT INTO curso
  SELECT curso_cod, nombre, 'FIN', creditos
        FROM cursosNuevos
        WHERE nombre_unid = 'FIN';
```

Actualizaciones (updates)

- Se pueden actualizar los valores de un atributo para todos los registro

```
UPDATE curso
SET creditos = creditos+1;
```

Actualizaciones (updates)

- Se pueden actualizar los valores de un atributo para todos los registro

```
UPDATE curso
SET creditos = creditos+1;
```

- Se pueden actualizar solo algunos registros que cumplan con un predicado

```
UPDATE curso
SET creditos = creditos+1
WHERE creditos < 3;
```

Actualizaciones (updates)

- Se pueden actualizar los valores de un atributo para todos los registro

```
UPDATE curso
SET creditos = creditos+1;
```

- Se pueden actualizar solo algunos registros que cumplan con un predicado

```
UPDATE curso
SET creditos = creditos+1
WHERE creditos < 3;
```

- Otro ejemplo de actualización parcial de acuerdo a un subconsulta

```
UPDATE curso
SET creditos = creditos+1
WHERE creditos < (
  SELECT avg(creditos)
         FROM CURSO
         );
```

Actualizaciones (updates)

- Se pueden considerar varios casos en la actualización

```
UPDATE curso
SET creditos =
  CASE
    WHEN creditos < 3 THEN creditos+1
          ELSE creditos - 1
        END;
```

Actualizaciones (updates)

- Se pueden considerar varios casos en la actualización

```
UPDATE curso
SET creditos =
  CASE
    WHEN creditos < 3 THEN creditos+1
          ELSE creditos - 1
        END;
```

- El orden en que se realizan updates es importante ya que se están modificando registros, y estos pueden afectar el resultado de los predicados evaluados.

Vistas

Vistas (views)

- Algunos usuarios requieren acceso a cierta información, pero no necesariamente a toda la base de datos.

Vistas (views)

- Algunos usuarios requieren acceso a cierta información, pero no necesariamente a toda la base de datos.
- Es deseable que la información aparezca en forma de una relación/tabla sin que necesariamente sea parte del esquema de datos (atributos de varias relaciones)

Vistas (views)

- Algunos usuarios requieren acceso a cierta información, pero no necesariamente a toda la base de datos.
- Es deseable que la información aparezca en forma de una relación/tabla sin que necesariamente sea parte del esquema de datos (atributos de varias relaciones)
- Ejemplo: listado de todos los cursos, sus estudiantes y sus notas

```
SELECT nombres, apellidos, nombre as curso, nota FROM
        (SELECT *
        FROM estudiante join estCursos
        ON estudiante.id = estCursos.estID) as A
        JOIN curso
        ON A.curso_cod = curso.curso_cod;
```

Vistas (views)

- Vista: relación construida a partir de una consulta

Vistas (views)

- Vista: relación construida a partir de una consulta
- La vista se mantiene actualizada

Vistas (views)

- Vista: relación construida a partir de una consulta
- La vista se mantiene actualizada
- La consulta se ejecuta cada vez que se accede a la vista

Vistas (views)

- Vista: relación construida a partir de una consulta
- La vista se mantiene actualizada
- La consulta se ejecuta cada vez que se accede a la vista
- Comando `CREATE VIEW nombre_vista AS`

```
CREATE VIEW resumenCursosNotas AS
SELECT nombres, apellidos, nombre as curso, nota FROM
        (SELECT *
        FROM estudiante join estCursos
        ON estudiante.id = estCursos.estID) as A
        JOIN curso
        ON A.curso_cod = curso.curso_cod;
```

Vistas (views)

- Las vistas se pueden usar para realizar consultas

Vistas (views)

- Las vistas se pueden usar para realizar consultas
- Listar estudiantes con notas sobresalientes

```
SELECT nombres, apellidos, nombre as curso
FROM resumenCursosNotas
WHERE nota > 4.5;
```

Vistas (views)

- Las vistas se pueden usar para realizar consultas
- Listar estudiantes con notas sobresalientes

```
SELECT nombres, apellidos, nombre as curso
FROM resumenCursosNotas
WHERE nota > 4.5;
```

- Una vista puede usarse para definir consultas que definan otras vistas

Vistas (views)

- Vistas materializadas: se crea efectivamente la tabla y se almacena en disco

```
CREATE MATERIALIZED VIEW nombreVistaMat AS
SELECT ...;
```

Vistas (views)

- Vistas materializadas: se crea efectivamente la tabla y se almacena en disco
- Se debe actualizar cuando cambie la información en las tablas originales

```
CREATE MATERIALIZED VIEW nombreVistaMat AS
SELECT ...;
```

Vistas (views)

- Vistas materializadas: se crea efectivamente la tabla y se almacena en disco
- Se debe actualizar cuando cambie la información en las tablas originales
- Útil al tener muchas consultas que deben responderse rápidamente (e.g., servicios web)

```
CREATE MATERIALIZED VIEW nombreVistaMat AS
SELECT ...;
```

Vistas (views)

- Vistas materializadas: se crea efectivamente la tabla y se almacena en disco
- Se debe actualizar cuando cambie la información en las tablas originales
- Útil al tener muchas consultas que deben responderse rápidamente (e.g., servicios web)
- Mantenimiento requiere cuidado (más adelante)

```
CREATE MATERIALIZED VIEW nombreVistaMat AS
SELECT ...;
```

Vistas (views)

- El usuario de la vista podría querer actualizarla, por ejemplo insertar o modificar registros

Vistas (views)

- El usuario de la vista podría querer actualizarla, por ejemplo insertar o modificar registros
- Por ejemplo, agregar un registro a `resumenCursosNotas`

Vistas (views)

- El usuario de la vista podría querer actualizarla, por ejemplo insertar o modificar registros
- Por ejemplo, agregar un registro a `resumenCursosNotas`
- Esto puede generar problemas ya que requiere una serie de inserciones/actualizaciones/eliminaciones en las tablas originales (agregar un estudiante, agregar el curso, agregar la nota)

Vistas (views)

- El usuario de la vista podría querer actualizarla, por ejemplo insertar o modificar registros
- Por ejemplo, agregar un registro a `resumenCursosNotas`
- Esto puede generar problemas ya que requiere una serie de inserciones/actualizaciones/eliminaciones en las tablas originales (agregar un estudiante, agregar el curso, agregar la nota)
- Una vista es actualizable si la consulta con la que se construye:
  1. Se refiere a una única relación en la cláusula `FROM`

Vistas (views)

- El usuario de la vista podría querer actualizarla, por ejemplo insertar o modificar registros
- Por ejemplo, agregar un registro a `resumenCursosNotas`
- Esto puede generar problemas ya que requiere una serie de inserciones/actualizaciones/eliminaciones en las tablas originales (agregar un estudiante, agregar el curso, agregar la nota)
- Una vista es actualizable si la consulta con la que se construye:
  1. Se refiere a una única relación en la cláusula `FROM`
  2. La cláusula `SELECT` tiene solo atributos (no hay expresiones, agregados, o distinct)

Vistas (views)

- El usuario de la vista podría querer actualizarla, por ejemplo insertar o modificar registros
- Por ejemplo, agregar un registro a `resumenCursosNotas`
- Esto puede generar problemas ya que requiere una serie de inserciones/actualizaciones/eliminaciones en las tablas originales (agregar un estudiante, agregar el curso, agregar la nota)
- Una vista es actualizable si la consulta con la que se construye:
  1. Se refiere a una única relación en la cláusula `FROM`
  2. La cláusula `SELECT` tiene solo atributos (no hay expresiones, agregados, o distinct)
  3. Todo atributo no mencionado en la cláusula `SELECT` puede asignársele NULL

Vistas (views)

- El usuario de la vista podría querer actualizarla, por ejemplo insertar o modificar registros
- Por ejemplo, agregar un registro a `resumenCursosNotas`
- Esto puede generar problemas ya que requiere una serie de inserciones/actualizaciones/eliminaciones en las tablas originales (agregar un estudiante, agregar el curso, agregar la nota)
- Una vista es actualizable si la consulta con la que se construye:
  1. Se refiere a una única relación en la cláusula `FROM`
  2. La cláusula `SELECT` tiene solo atributos (no hay expresiones, agregados, o distinct)
  3. Todo atributo no mencionado en la cláusula `SELECT` puede asignársele NULL
  4. No tiene cláusulas `GROUP BY` o `HAVING`

Transacciones

Transacciones

- Transacción: sucesión de comandos de consulta o modificación que conforman una unidad

Transacciones

- Transacción: sucesión de comandos de consulta o modificación que conforman una unidad
- Una transacción es atómica: indivisible

Transacciones

- Transacción: sucesión de comandos de consulta o modificación que conforman una unidad
- Una transacción es atómica: indivisible
- La transacción termina con

Transacciones

- Transacción: sucesión de comandos de consulta o modificación que conforman una unidad
- Una transacción es atómica: indivisible
- La transacción termina con
  - **Commit** (work): todas las modificaciones realizadas en al transacción se vuelven permanentes

Transacciones

- Transacción: sucesión de comandos de consulta o modificación que conforman una unidad
- Una transacción es atómica: indivisible
- La transacción termina con
  - **Commit** (work): todas las modificaciones realizadas en al transacción se vuelven permanentes
  - **Rollback** (work): se deshacen todos los cambios hechos durante la transacción y se regresa la base de datos al estado anterior a la ejecución de la transacción

Transacciones

- En el estándar SQL:1999 las transacciones se declaran con los comandos `begin atomic ... end`

Transacciones

- En el estándar SQL:1999 las transacciones se declaran con los comandos `begin atomic ... end`
- En PostgreSQL se usan los comandos `BEGIN ... COMMIT`

Transacciones

- En el estándar SQL:1999 las transacciones se declaran con los comandos `begin atomic ... end`
- En PostgreSQL se usan los comandos `BEGIN ... COMMIT`
- Ejemplo:

```
BEGIN;
INSERT INTO estudiante VALUES('Juan', 'Perez', 8888);
INSERT INTO estCursos VALUES(8888, '000', 4.2);
COMMIT;
```

Restricciones de Integridad

Restricciones de Integridad - Una relación

- Atributos no nulos

Restricciones de Integridad - Una relación

- Atributos no nulos

- `not null`

```
create table estudiante
  (nombres varchar (20) not null,
  apellidos varchar (20) not null,
  id int,
  primary key (id)
  );
```

Restricciones de Integridad - Una relación

- Atributos únicos (llaves primarias candidatas)

Restricciones de Integridad - Una relación

- Atributos únicos (llaves primarias candidatas)

- `unique`

```
create table estudiante
  (nombres varchar (20) not null,
  apellidos varchar (20) not null,
  id int,
        unique (nombres, apellidos),
  primary key (id)
  );
```

Restricciones de Integridad - Una relación

- Atributos que deben cumplir algunas restricciones

Restricciones de Integridad - Una relación

- Atributos que deben cumplir algunas restricciones

- check

```
create table estudiante
  (nombres varchar (20) not null,
  apellidos varchar (20) not null,
  id int,
        check (id between 1 and 10000000000),
  primary key (id)
  );
```

Restricciones de Integridad - Varias relaciones

- Integralidad por referencia

```
create table unid_acad
  (nombre_unid varchar (20),
  edificio varchar (15) not null,
  presupuesto numeric (12,2),
  primary key (nombre_unid));

create table instructor
  (inst_ID varchar (5),
  nombre varchar (20) not null,
  nombre_unid varchar (20),
  salario numeric (8,2),
  primary key (inst_ID),
  foreign key (nombre_unid) references unid_acad);
```

Restricciones de Integridad - Varias relaciones

- Integralidad por referencia
- LLaves foráneas

```
create table unid_acad
  (nombre_unid varchar (20),
  edificio varchar (15) not null,
  presupuesto numeric (12,2),
  primary key (nombre_unid));

create table instructor
  (inst_ID varchar (5),
  nombre varchar (20) not null,
  nombre_unid varchar (20),
  salario numeric (8,2),
  primary key (inst_ID),
  foreign key (nombre_unid) references unid_acad);
```

Restricciones de Integridad - Varias relaciones

- ¿Qué hacer si cambia la referencia?

```
create table instructor
  (inst_ID varchar (5),
  nombre varchar (20) not null,
  nombre_unid varchar (20),
  salario numeric (8,2),
  primary key (inst_ID),
  foreign key (nombre_unid) references unid_acad
          on delete cascade
          on update cascade
        );
```

Restricciones de Integridad - Varias relaciones

- ¿Qué hacer si cambia la referencia?
- Se generaría una violación a restricción

```
create table instructor
  (inst_ID varchar (5),
  nombre varchar (20) not null,
  nombre_unid varchar (20),
  salario numeric (8,2),
  primary key (inst_ID),
  foreign key (nombre_unid) references unid_acad
          on delete cascade
          on update cascade
        );
```

Restricciones de Integridad - Varias relaciones

- ¿Qué hacer si cambia la referencia?
- Se generaría una violación a restricción
- Por defecto: se cancela la operación

```
create table instructor
  (inst_ID varchar (5),
  nombre varchar (20) not null,
  nombre_unid varchar (20),
  salario numeric (8,2),
  primary key (inst_ID),
  foreign key (nombre_unid) references unid_acad
          on delete cascade
          on update cascade
        );
```

Restricciones de Integridad - Varias relaciones

- ¿Qué hacer si cambia la referencia?
- Se generaría una violación a restricción
- Por defecto: se cancela la operación
- Alternativa

```
create table instructor
  (inst_ID varchar (5),
  nombre varchar (20) not null,
  nombre_unid varchar (20),
  salario numeric (8,2),
  primary key (inst_ID),
  foreign key (nombre_unid) references unid_acad
          on delete cascade
          on update cascade
        );
```

Restricciones de Integralidad - Varias relaciones

- `on update cascade`

Restricciones de Integralidad - Varias relaciones

- `on update cascade`
- `on update set null`

Restricciones de Integralidad - Varias relaciones

- `on update cascade`
- `on update set null`
- `on update set default`

Aserciones

- Condiciones complejas a revisar cuando se realizan cambios

Aserciones

- Condiciones complejas a revisar cuando se realizan cambios
- No caben en la cláusula `check`

Aserciones

- Condiciones complejas a revisar cuando se realizan cambios
- No caben en la cláusula `check`
- No se implementan en muchas distribuciones

```
CREATE ASSERTION nombre
CHECK condicion;
```

Más sobre Tipos de Datos

Fechas

- Fecha (date): 'yyyy-mm-dd'

Fechas

- Fecha (date): 'yyyy-mm-dd'
- Hora (time): 'HH:MM:SS.xx'

Fechas

- Fecha (date): 'yyyy-mm-dd'
- Hora (time): 'HH:MM:SS.xx'
- Timestamp: 'yyyy-mm-dd HH:MM:SS.xx'

Fechas

- Fecha (date): 'yyyy-mm-dd'
- Hora (time): 'HH:MM:SS.xx'
- Timestamp: 'yyyy-mm-dd HH:MM:SS.xx'
- Extraer partes de la fecha

```
SELECT EXTRACT(YEAR FROM TIMESTAMP '2018-11-31␣
15:20:15');
```

Fechas

- Fecha (date): 'yyyy-mm-dd'
- Hora (time): 'HH:MM:SS.xx'
- Timestamp: 'yyyy-mm-dd HH:MM:SS.xx'
- Extraer partes de la fecha

```
SELECT EXTRACT(YEAR FROM TIMESTAMP '2018-11-31␣
15:20:15');
```

- Operaciones: SELECT DATE '2018-11-31' - DATE '2016-21-3');
- Y muchas mas: `https://www.postgresql.org/docs/9.1/functions-datetime.html`

Índices

- Índice: permite realizar búsquedas basadas en los valores de un atributo rápidamente

Índices

- Índice: permite realizar búsquedas basadas en los valores de un atributo rápidamente
- Ejemplo:

```
CREATE UNIQUE INDEX estID_ind ON estudiante (id);
```

Índices

- Índice: permite realizar búsquedas basadas en los valores de un atributo rápidamente
- Ejemplo:

```
CREATE UNIQUE INDEX estID_ind ON estudiante (id);
```

- Unique: revisa que no haya repetidos

Objetos de gran tamaño

- `clob`: character large object

Objetos de gran tamaño

- `clob`: character large object
- `blob`: binary large object

Objetos de gran tamaño

- `clob`: character large object
- `blob`: binary large object
- Ejemplo:

```
create table instructor
  (inst_ID varchar (5),
  nombres varchar (20) not null,
  apellidos varchar (20) not null,
  nombre_unid varchar (20),
  foto blob(10MB)
       cv clob(500KB)
       video blob(10GB);
```

Tipos definidos por el usuario

- `distinc types`: tipos definidos que permiten comparaciones y operaciones en ellos, pero no con otros

Tipos definidos por el usuario

- `distinc types`: tipos definidos que permiten comparaciones y operaciones en ellos, pero no con otros

- Ejemplo:

```
create table instructor
  (inst_ID varchar (5),
  nombres varchar (20) not null,
  apellidos varchar (20) not null,
  nombre_unid varchar (20)
        salario numeric (10,2);
```

Tipos definidos por el usuario

- Crear tipo:

```
create type pesos as numeric (10,2);
create type dolares as numeric (7,2);
```

Tipos definidos por el usuario

- Crear tipo:

```
create type pesos as numeric (10,2);
create type dolares as numeric (7,2);
```

- Usar tipo:

```
create table instructor
  (inst_ID varchar (5),
  nombres varchar (20) not null,
  apellidos varchar (20) not null,
  nombre_unid varchar (20)
        salario pesos;
```

Tipos definidos por el usuario

- Si se quiere comparar dos tipos diferentes u operar sobre ellos es necesario un casting:

```
CAST (instructor.salario to numeric(12,2) )
```

Tipos definidos por el usuario: dominios

- Dominio incluye restricciones y valores por defecto.

Tipos definidos por el usuario: dominios

- Dominio incluye restricciones y valores por defecto.
- Dominio no es de tipo fuerte, luego se puede comparar con otros tipos compatibles

Tipos definidos por el usuario: dominios

- Dominio incluye restricciones y valores por defecto.
- Dominio no es de tipo fuerte, luego se puede comparar con otros tipos compatibles
- Definir dominio:

```
CREATE DOMAIN nombre_de_contacto AS
    VARCHAR(20) NOT NULL CHECK (value !~ '\s');
```

Tipos definidos por el usuario: dominios

- Dominio incluye restricciones y valores por defecto.
- Dominio no es de tipo fuerte, luego se puede comparar con otros tipos compatibles
- Definir dominio:

```
CREATE DOMAIN nombre_de_contacto AS
    VARCHAR(20) NOT NULL CHECK (value !~ '\s');
```

- Usar dominio:

```
create table instructor
  (inst_ID varchar (5),
  nombres nombre_de_contacto,
  apellidos nombre_de_contacto,
  nombre_unid varchar (20)
  salario pesos;
```

Crear tablas a partir de otras

- Se puede crear nuevas tablas usando el esquema de una existente

```
create table curso2
  (like curso);
```

Crear tablas a partir de otras

- Se puede crear nuevas tablas usando el esquema de una existente

```
create table curso2
  (like curso);
```

- Se puede crear nuevas tablas usando el esquema y los datos de una existente

```
create table curso2 as
  (select *
        from curso
        where unid_acad = 'MACC')
with data;
```

Autorización de acceso

Autorización de acceso a la base de datos

- A cada usuario o rol de usuario se le definen privilegios

Autorización de acceso a la base de datos

- A cada usuario o rol de usuario se le definen privilegios
- Privilegios de lectura, inserción, actualización o eliminación

Autorización de acceso a la base de datos

- A cada usuario o rol de usuario se le definen privilegios
- Privilegios de lectura, inserción, actualización o eliminación
- Lectura:

```
CREATE USER luis WITH PASSWORD '*w3e!5';

GRANT SELECT ON instructor TO luis;
```

Autorización de acceso a la base de datos

- Actualización:

```
GRANT UPDATE (salario) ON instructor TO luis, ramon;
```

Autorización de acceso a la base de datos

- Actualización:

```
GRANT UPDATE (salario) ON instructor TO luis, ramon;
```

- Inserción:

```
GRANT INSERT (id, nombres, apellidos) ON instructor TO
    luis, ramon;
```

Autorización de acceso a la base de datos

- Actualización:

```
GRANT UPDATE (salario) ON instructor TO luis, ramon;
```

- Inserción:

```
GRANT INSERT (id, nombres, apellidos) ON instructor TO
    luis, ramon;
```

- Eliminación:

```
GRANT DELETE ON instructor TO ramon;
```

Autorización de acceso a la base de datos

- Quitar autorizaciones:

```
REVOKE UPDATE (salario) ON instructor FROM luisS;
```

Roles

- Grupos de usuarios con los mismos privilegios

```
CREATE ROLE sec_academico;
GRANT UPDATE (salario) ON instructor TO sec_academico;
```

Roles

- Grupos de usuarios con los mismos privilegios

```
CREATE ROLE sec_academico;
GRANT UPDATE (salario) ON instructor TO sec_academico;
```

- De rol a rol

```
CREATE ROLE decano;
GRANT sec_academico TO decano;
```

Roles

- Grupos de usuarios con los mismos privilegios

```
CREATE ROLE sec_academico;
GRANT UPDATE (salario) ON instructor TO sec_academico;
```

- De rol a rol

```
CREATE ROLE decano;
GRANT sec_academico TO decano;
```

- De rol a usuario

```
GRANT decano TO ramiro;
```

Roles

- Grupos de usuarios con los mismos privilegios

```
CREATE ROLE sec_academico;
GRANT UPDATE (salario) ON instructor TO sec_academico;
```

- De rol a rol

```
CREATE ROLE decano;
GRANT sec_academico TO decano;
```

- De rol a usuario

```
GRANT decano TO ramiro;
```

- Todos los privilegios

```
GRANT ALL PRIVILEGES ON instructor TO decano;
```

Roles y Autorizaciones

- Un usuario puede que tiene un privilegio puede autorizar a otro con el mismo privilegio

```
GRANT UPDATE (salario) ON instructor TO ramiro WITH
    GRANT OPTION;
```

Roles y Autorizaciones

- Un usuario puede que tiene un privilegio puede autorizar a otro con el mismo privilegio

```
GRANT UPDATE (salario) ON instructor TO ramiro WITH
    GRANT OPTION;
```

- Si 'ramiro' pierde sus privilegios, también los pierde todo usuario que los haya recibido de 'ramiro'

Roles y Autorizaciones

- Un usuario puede que tiene un privilegio puede autorizar a otro con el mismo privilegio

```
GRANT UPDATE (salario) ON instructor TO ramiro WITH
    GRANT OPTION;
```

- Si 'ramiro' pierde sus privilegios, también los pierde todo usuario que los haya recibido de 'ramiro'

- Este comportamiento se puede prevenir

```
REVOKE UPDATE (salario) ON instructor FROM ramiro
    RESTRICT;
```

Roles y Autorizaciones

- Un usuario puede que tiene un privilegio puede autorizar a otro con el mismo privilegio

```
GRANT UPDATE (salario) ON instructor TO ramiro WITH
    GRANT OPTION;
```

- Si 'ramiro' pierde sus privilegios, también los pierde todo usuario que los haya recibido de 'ramiro'

- Este comportamiento se puede prevenir

```
REVOKE UPDATE (salario) ON instructor FROM ramiro
    RESTRICT;
```

- En este caso no se realiza el `revoke` is 'ramiro' ha autorizado otros usuarios