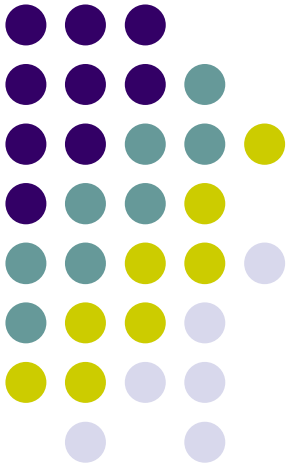
# Bases de datos

## Tema III – El lenguaje estructurado de consulta - SQL

Universidad Nacional del
FACULTAD DE INGEN
Y CIENCIAS HÍDRIC



# RDBMS *Relational DataBase Management System*

## Principios de la RDBMS

**Regla 0: Gestión de una base de datos relacional.**

**Todo sistema que se anuncie como un sistema de gestión de base de datos relacional, debe ser capaz de manejar bases de datos exclusivamente con sus capacidades relacionales.**

**Regla 1: Representación de la información.**

**Toda la información de una base de datos relacional, se representa explícitamente en el ámbito lógico y exactamente de una forma: mediante valores en tablas.**

**Regla 2: Garantía de accesibilidad lógica.**

**Todos y cada uno de los datos de una base de datos, relacional tienen la garantía de ser accesibles lógicamente mediante el recurso de una combinación de: el nombre de la Tabla, el valor de la clave primaria y el nombre de la columna.**

**Regla 3: Representación sistemática de la información que falta.**

**Los valores nulos (que son distintos de la cadena vacía de caracteres o de la cadena de caracteres en blanco, y distintos de cero o de cualquier otro número) tienen la existencia en los sistemas de gestión de bases de datos totalmente relacionales, para representar la información que falta y la información que no es aplicable, de forma sistemática e independiente del tipo de dato.**

# RDBMS *Relational DataBase Management System*

## Principios de la RDBMS

**Regla 4: Sub-lenguaje de datos completo.**

**Un sistema relacional puede soportar varios lenguajes y varios modos de uso terminal. Sin embargo, debe haber, al menos, un lenguaje cuyas instrucciones puedan expresarse por alguna sintaxis bien definida, como cadenas de caracteres, y que sea completo, soportando todos los términos siguientes:**

**Definición de Datos**

**Definición de Vistas**

**Manejo de Datos**

**Limitaciones de integridad**

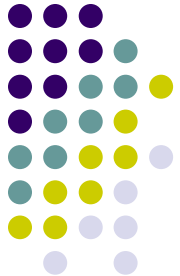
**Autorización o permisos**

**Límites de transacción (inicio y fin para hacer permanentes los cambios y deshacer los cambios no permanentes)**

**Regla 5: Inserción, actualización y borrado de alto nivel.**

**La capacidad de manejar una relación de base o una relación derivada como un único operador, se aplica no sólo a la recuperación de datos, sino también a la inserción, a la actualización y al borrado de datos.**

# RDBMS *Relational DataBase Management System*



## Principios de la RDBMS

**Regla 6: Independencia de los datos físicos.**

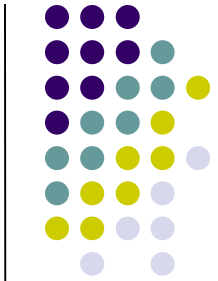
**Los programas de aplicaciones y las actividades terminales, permanecerán lógicamente inalterados siempre que se realicen cambios en las representaciones de almacenamiento o en los métodos de acceso.**

**Regla 7: Independencia de los datos lógicos.**

**Los programas de aplicaciones y las actividades finales permanecerán lógicamente inalterados cuando se llevan a cabo cambios en las tablas de base que conservan la información de cualquier tipo que permita teóricamente su inalterabilidad.**

**Regla 8: Independencia de la integridad.**

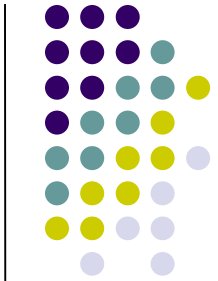
**Las limitaciones de integridad, específicas de una base de datos en particular, deben ser definibles en un sub-lenguaje de definición de datos y almacenables en el catálogo o diccionario.**



# Componentes del SQL

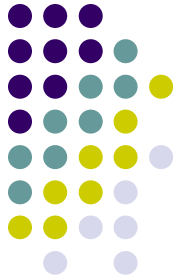
- **Catálogo o diccionario de datos**
- **Los comandos**
- **Las palabras reservadas**
- **Los conectores lógicos y predicados**
- **El lenguaje de definición de datos**
- **El lenguaje de manejo de datos**
- **El lenguaje de control de datos**

# SQL *Structured Query Language*



## Comandos del SQL

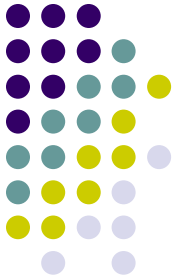
**ALTER TABLE**
**BEGIN TRANSACTION**
**COMMIT TRANSACTION**
**CREATE**
**DELETE**
**DROP**
**GRANT**
**INSERT**
**REVOKE**
**ROLLBACK**
**SELECT**
**UPDATE**



# Las palabras reservadas

**Son las palabras claves que forman los comandos de SQL, así como a las palabras calificativas que se usan con ellos.**

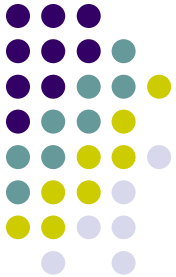
**No pueden utilizarse como identificadores o sea que no se pueden usar como nombres de tablas, de vistas o de columnas sin aplicarles símbolos especiales definidos por el implementador, con el fin de que se distingan de sus correspondientes palabras claves.**



# Las palabras reservadas

**Son las palabras claves que forman los comandos de SQL, así como a las palabras calificativas que se usan con ellos.**

**No pueden utilizarse como identificadores o sea que no se pueden usar como nombres de tablas, de vistas o de columnas sin aplicarles símbolos especiales definidos por el implementador, con el fin de que se distingan de sus correspondientes palabras claves.**



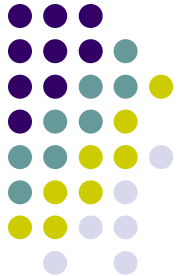
# Tipos de datos y operadores

**SQL funciona con tres tipos principales datos:**

- **Cadenas alfanuméricas (CHAR, CHARACTER VARING)**
- **Numérico**
  - **Entero**
    - **Cortos (byte, smallint, short)**
    - **Largos (integer, bigint, numeric)**
  - **Decimal**
- **Fecha (DATE)**

**Operaciones básicas de valores:**

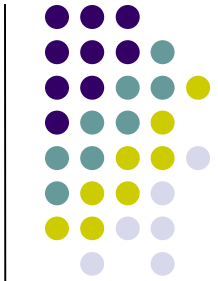
| | |
|---|---|
| **Suma** | **(+)** |
| **Resta** | **(-)** |
| **Multiplicación** | **(*)** |
| **División** | **(/)** |



# Lenguaje de definición de datos

**Es aquel que sus instrucciones interactúan con el diccionario de datos manipulando su contenido agregando, modificando o eliminando su información. Los comandos son:**

**CREATE …**
**ALTER …**
**DROP …**



# Lenguaje de definición de datos

***CREATE DATABASE nombre_de_la_base [adicionales]***

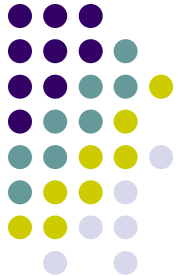
**Dentro de adicionales, pueden ir distintos elementos que dependerán del motor sobre el que se ejecute la orden. Estos elementos pueden ser:**

- **si llevará o no archivo de *log*;**
- **espacio o dispositivo donde se alojarán los datos,**
- **espacio o dispositivo donde se alojará el log,**
- **espacio inicial tanto para datos como para log,**
- **forma de crecimiento de los dispositivos,**
- **etc**

***CREATE SCHEMA nombre_del_esquema***

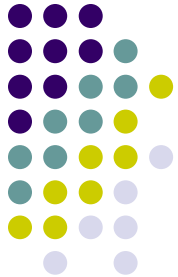
# Lenguaje de definición de datos

```
CREATE DATABASE hospital;

CREATE SCHEMA persona;
/* ahi pondremos las tablas
provincia, localidad, persona, medico, empleado */

CREATE SCHEMA estructura;
/* ahi pondremos las tablas seccion, sector, sala */

CREATE SCHEMA gestion;
/* ahi pondremos las tablas historial, especialidad, trabaja_en, asignacion */
```



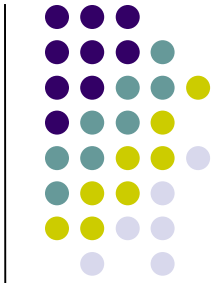
# Lenguaje de definición de datos

```
CREATE TABLE [esquema.]nombre_tabla (
    Columna1  tipo_dato (tamaño)
        [NOT NULL] [DEFAULT valor_por_default]
        [CONSTRAINT nombre_constraint_de_columna]
        [UNIQUE] [PRIMARY KEY]
        [REFERENCES [esquema.]nombre_tabla (columna/s)]
        [CHECK (condición)],
    …..,  ColumnaN….,
    [CONSTRAINT nombre_constraint_de_tabla]
    [UNIQUE (columna/s)]
    [PRIMARY KEY (columna/s)]
    [CHECK (condición)]
    [FOREIGN KEY (columna/s) REFERENCES [esquema.]nombre_tabla
    (columna/s)]
);
```



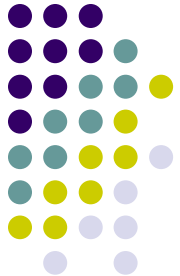
# Lenguaje de definición de datos

```
CREATE TABLE persona.provincia (
   id_provincia        smallint       not null,
   nom_provincia       varchar(30)  not null);

CREATE TABLE persona.localidad (
   id_provincia        smallint            not null,
   id_localidad        smallint            not null,
   nom_localidad       varchar(40)         not null);
```



# Lenguaje de definición de datos

```
CREATE TABLE persona.persona (
    tipodoc         char            not null,
    nrodoc          integer         not null,
    sexo            char            not null,
    apenom          varchar(40)     not null,
    domicilio       varchar(50)     null,
    fenaci              date            null,
    id_provivive    smallint        not null,
    id_locavive     smallint        not null,
    id_provinace    smallint        null,
    id_locanace     smallint        null,
    tipodocpadre    char            null,
    nrodocpadre     integer         null,
    sexopadre       char            null,
    tipodocmadre    char            null,
    nrodocmadre     integer         null,
    sexomadre       char            null);
```



# Lenguaje de definición de datos

***Conceptos asociados***

***Páginas***

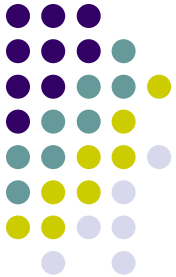
***Unidad básica de almacenamiento.***
***2048 bytes (2 Kb)***

***Densidad de filas (fillfactor, max_row_per_page)***

***Espacio utilizable de una página divido el tamaño de la fila***
***Porcentaje permitido para ocupación de una página***
***Máxima cantidad de filas por página***

***Extents***

***Espacio reservado simultáneamente para alojar objetos***
***8 páginas (16 Kb)***

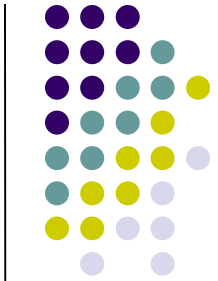


# Lenguaje de manejo de datos

## Inserción

```
INSERT INTO [esquema.]nombre_de_la_tabla
        [(nombre_de_columna1,
         nombre_de_columna2, ...)]
        VALUES ('valor1', 'valor2', ...);

INSERT INTO [esquema.]nombre_de_la_tabla
        [(nombre_de_columna1,
         nombre_de_columna2, ...)]
        (subconsulta);
```

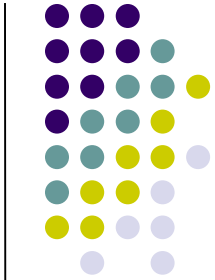
***Los valores que se inserten deberán ajustarse al tipo de datos de la columna en la que se van a insertar. Los valores CHAR y los tipo DATE deben ir entre comillas o apóstrofos mientras que los valores NUMERIC y NULL simplemente se indican.***



# Lenguaje de manejo de datos

## Inserción

```
INSERT INTO persona.provincia (id_provincia, nom_provincia)
    VALUES (1, 'SANTA FE');
INSERT INTO persona.provincia (id_provincia, nom_provincia)
    VALUES (2, 'ENTRE RIOS');

INSERT INTO persona.provincia VALUES (3, 'CORDOBA');
INSERT INTO persona.provincia VALUES (4, 'BUENOS AIRES');

INSERT INTO persona.localidad (id_provincia, id_localidad, nom_localidad)
    VALUES ( 1, 1, 'SANTA FE');

INSERT INTO persona.localidad VALUES ( 1, 2, 'SANTO TOME');
INSERT INTO persona.localidad VALUES ( 2, 1, 'PARANALANDIA');
INSERT INTO persona.localidad VALUES ( 2, 2, 'CONCORDIA');
```



# Lenguaje de manejo de datos

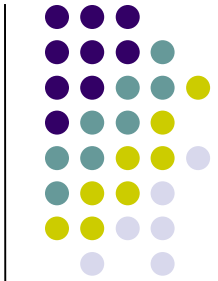
## Actualización

```
UPDATE  [esquema.]nombre_de_la_tabla
        SET     columna1 = valor_nuevo,
                columna2 = valor_nuevo,
                ...
                columnaN = valor_nuevo
        [WHERE condicion];
```

***La cláusula SET indica las columnas que hay que actualizar y qué valores hay que poner en ellas.***

***Actúa en todas las filas que satisfacen la condición especificada por la cláusula WHERE. La cláusula WHERE es opcional, pero si se omite, se actualizarán todas las filas de la tabla. Puede contener una subconsulta.***

***Se pueden actualizar múltiples columnas en cada fila, con un único comando UPDATE.***



# Lenguaje de manejo de datos

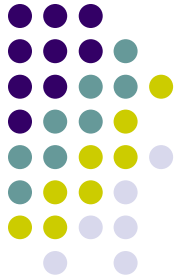
## Actualización

***Actualización por clave de negocio***

```
UPDATE persona.localidad SET nom_localidad = 'PARANA'
    WHERE id_provincia = 2 AND id_localidad = 1;
```

***Actualización por valor de atributo***

```
UPDATE persona.localidad SET nom_localidad = 'PARANA'
    Where nom_localidad LIKE 'PARANALANDIA%';
```



# Lenguaje de manejo de datos

## Borrado

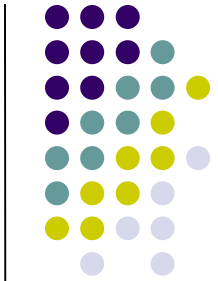
***DELETE FROM [esquema.]nombre_de_la_tabla***
***[WHERE condición];***

***No se pueden borrar parcialmente las filas.***

***La cláusula WHERE puede ser compleja y puede que incluya condiciones múltiples, conectores, y/o subconsultas.***

***Sin cláusula WHERE se suprimen todas las filas y dejará solamente las especificaciones de las columnas y el nombre de la tabla.***

***El comando TRUNCATE TABLE elimina todos los datos de la tabla indicada sin dejar rastros en el log, y consecuentemente no pueden deshacerse los cambios (el borrado). Es más rápido pero peligroso.***



# Lenguaje de manejo de datos

## Borrado

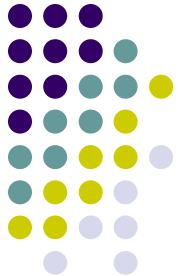
***Borrado por clave de negocio***

```
DELETE FROM persona.localidad
    WHERE id_provincia = 2 AND id_localidad = 1;
```

***Borrado por valor de atributo***

```
DELETE FROM persona.localidad
    WHERE nom_localidad LIKE 'PARANALANDIA%';
```

# SQL *Structured Query Language*



## Lenguaje de manejo de datos

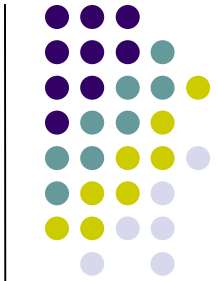
### Recuperación / Consulta

***SELECT columna1, columna2, ......,columnaN***
***FROM [esquema.]nombre_de_la_tabla;***

***Para recuperar solamente un conjunto de filas que se hayan especificado, se indicará la característica común de las filas que se quieren obtener mediante la cláusula WHERE.***

***La cláusula WHERE corresponde al predicado de selección del álgebra relacional. Consta de un predicado que incluye columnas de la tabla o tablas que aparecen en la cláusula FROM.***

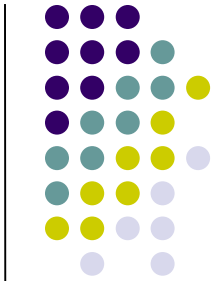
***WHERE es opcional, y en caso de omitirla, devuelve <u>todas</u> las filas de la tabla.***



# Lenguaje de manejo de datos

## Recuperación / Consulta

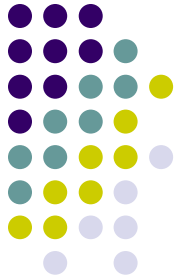
```
SELECT * FROM persona.localidad;
/*devuelve todas las columnas y filas */

SELECT nom_localidad FROM persona.localidad
        WHERE id_provincia = 1;
/*devuelve nombre de las localidades de la provincia 1*/

SELECT apenom, tipodoc, nrodoc FROM persona.persona
        WHERE id_provivive = 1 AND id_locavive = 1;
/*devuelve el nombre, tipo y número de documento de las personas que
viven en la provincia 1 (SANTA FE) y localidad 1 (SANTA FE) */
```



# Lenguaje de definición de datos

## Vistas

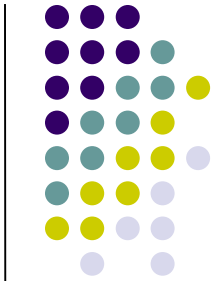
```
CREATE VIEW nombre_de_la_vista
      ((nombre_columnas_de_la_vista))
      AS
      (SELECT columna(s) FROM [esquema.]tabla(s) WHERE condicion(es));

CREATE VIEW persona_santafe
       (tipo_documento, nro_documento, apellido_nombre)
  AS (SELECT tipodoc, nrodoc, apenom FROM persona.persona
            WHERE id_provivive = 1 AND id_locavive = 1);
/* vista con tipo y número de documento y apellido y nombre de las
personas que viven en la provincial 1 y localidad 1) */
```



# Lenguaje de definición de datos

## Índices

- **Son estructuras de datos que permite acelerar la interacción con los datos de las tablas.**
- **Pueden crearse tantos índice como se deseen en una tabla cualquiera.**
- **Puede tener un índice para cada columna de la tabla, así como un índice para una combinación de columnas.**
- **Cuántos índices sean creados y de qué tipo para una determinada tabla, dependerán de los tipos de consultas esperadas que se dirigirán a la base de datos y además del tamaño de la misma.**
- **El crear demasiados índices puede presentar tantos inconvenientes como el crear muy pocos.**



# Lenguaje de definición de datos

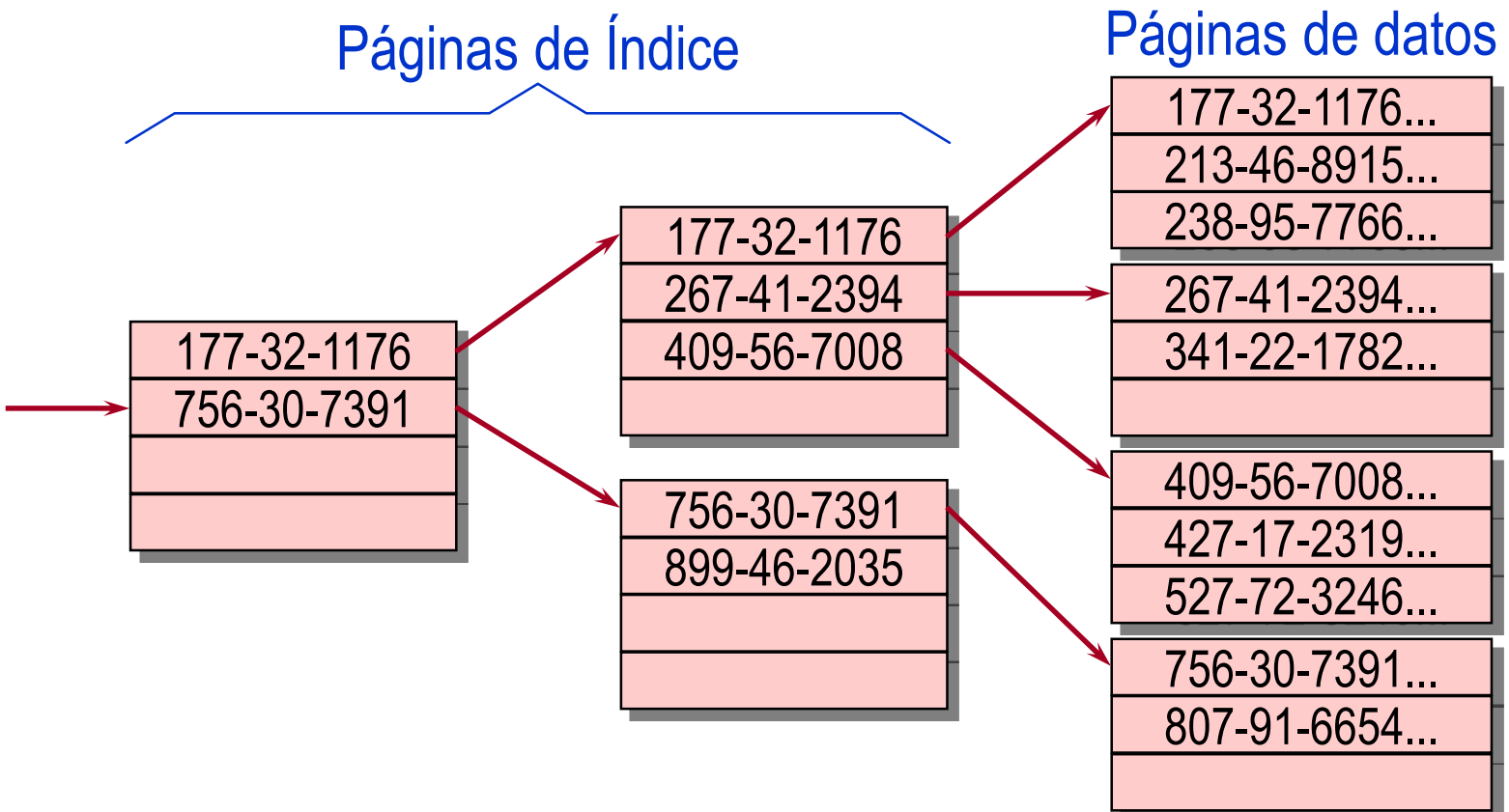
## Índices

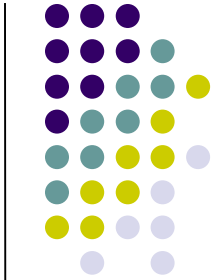
- **Mejoran el rendimiento reduciendo las entradas y salidas del disco. Realizan una función análoga a las de los índices de un libro acelerando la recuperación.**
- **Aseguran la unicidad. Puede crearse un índice único sobre una columna. Después, si se hace algún intento para insertar una fila que duplique el valor que ya está en esa columna, la inserción será rechazada. Por consiguiente, un UNIQUE INDEX actúa como una comprobación de la unicidad de la columna.**

# Lenguaje de definición de datos

## Índices

- Permite acceder directamente a filas de datos específicas.





# Lenguaje de definición de datos

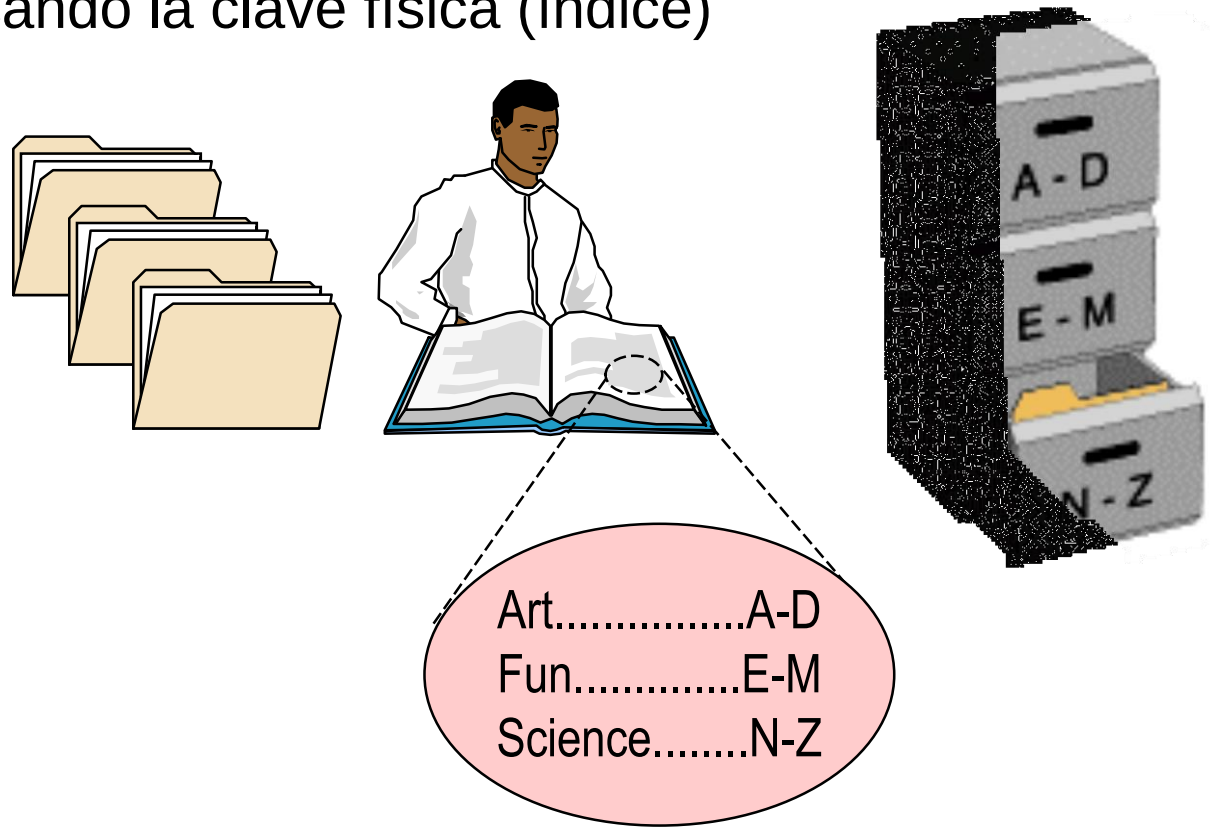
## Índices

- Existen dos tipos de índices:
  - Clustered
  - Nonclustered
- Una tabla puede tener:
  - Un máximo de un índice clustered
  - Un máximo de 249 índices nonclustered
- El orden físico de los datos depende del tipo de índice:
  - Si es un índice clustered, los datos estarán ordenados por la o las columnas indexadas
  - Si todos los índices son nonclustered o si no tiene índices, los datos estarán en el orden en el cual han sido insertados

# Lenguaje de definición de datos

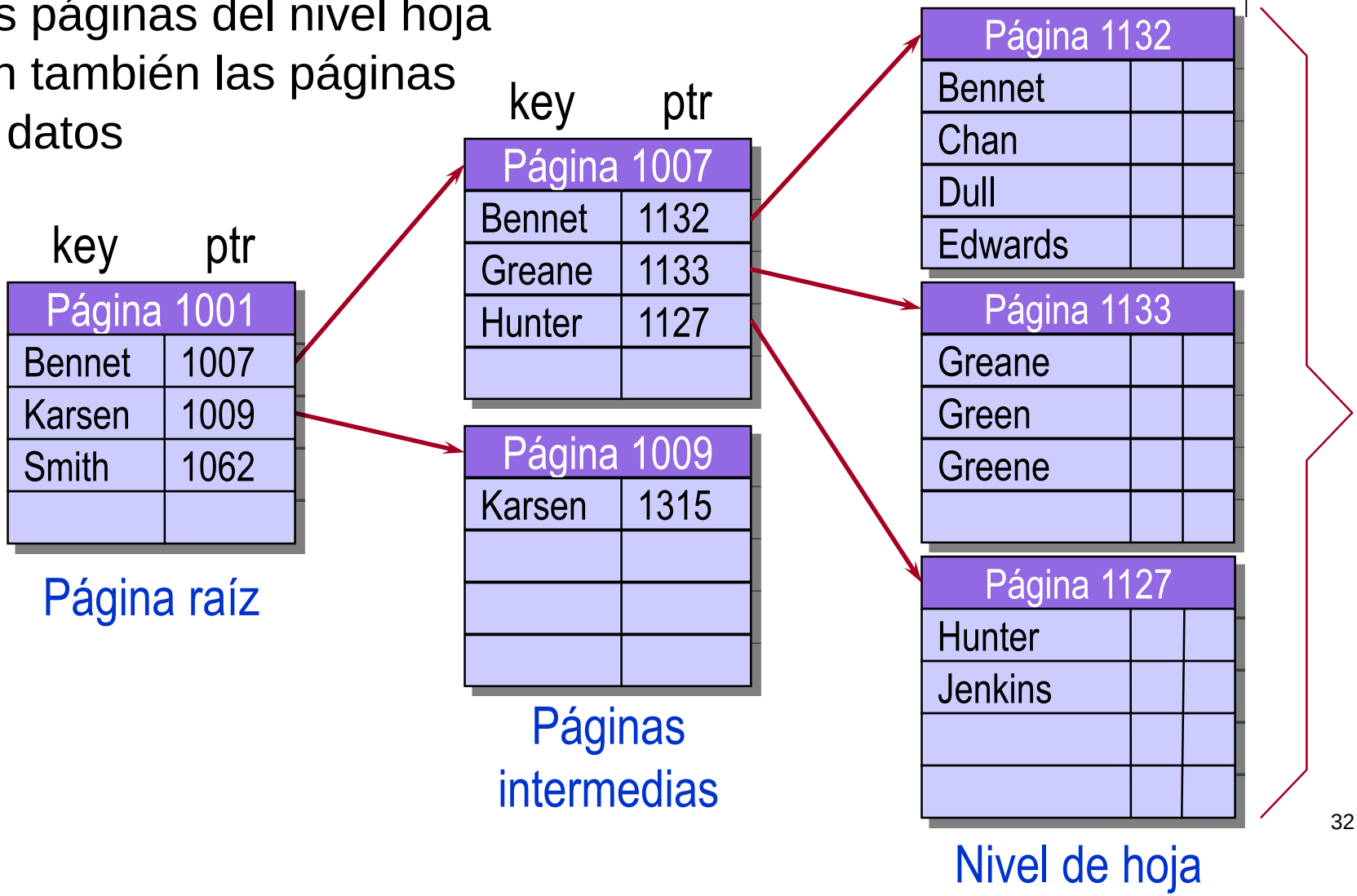
## Índices clustered

- Las filas de datos en la tabla están físicamente ordenadas
- Los usuarios definen cómo las filas de datos son ordenadas especificando la clave física (índice)



# Lenguaje de definición de datos

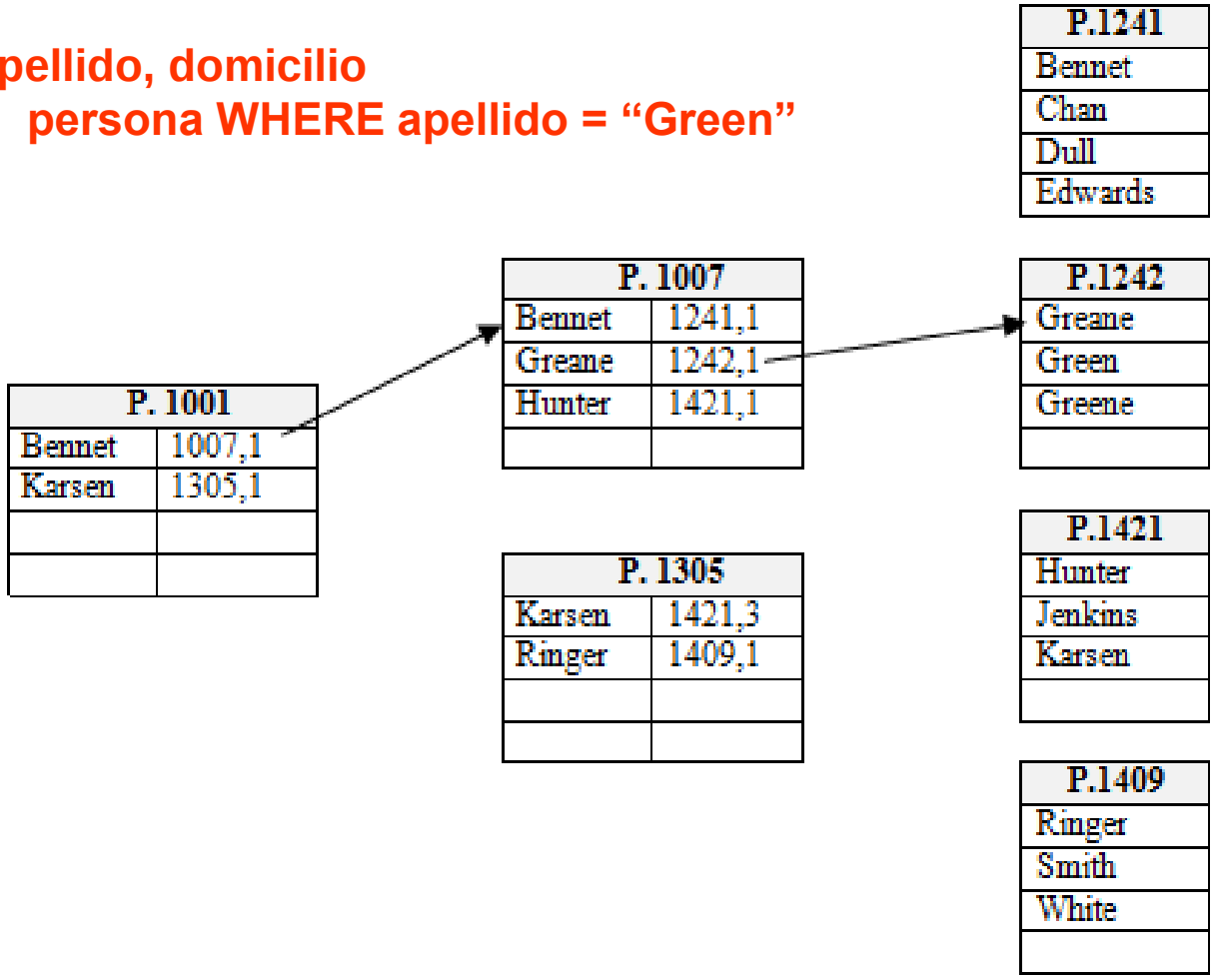
Las páginas del nivel hoja son también las páginas de datos



# SQL *Structured Query Language*

# Lenguaje de definición de datos

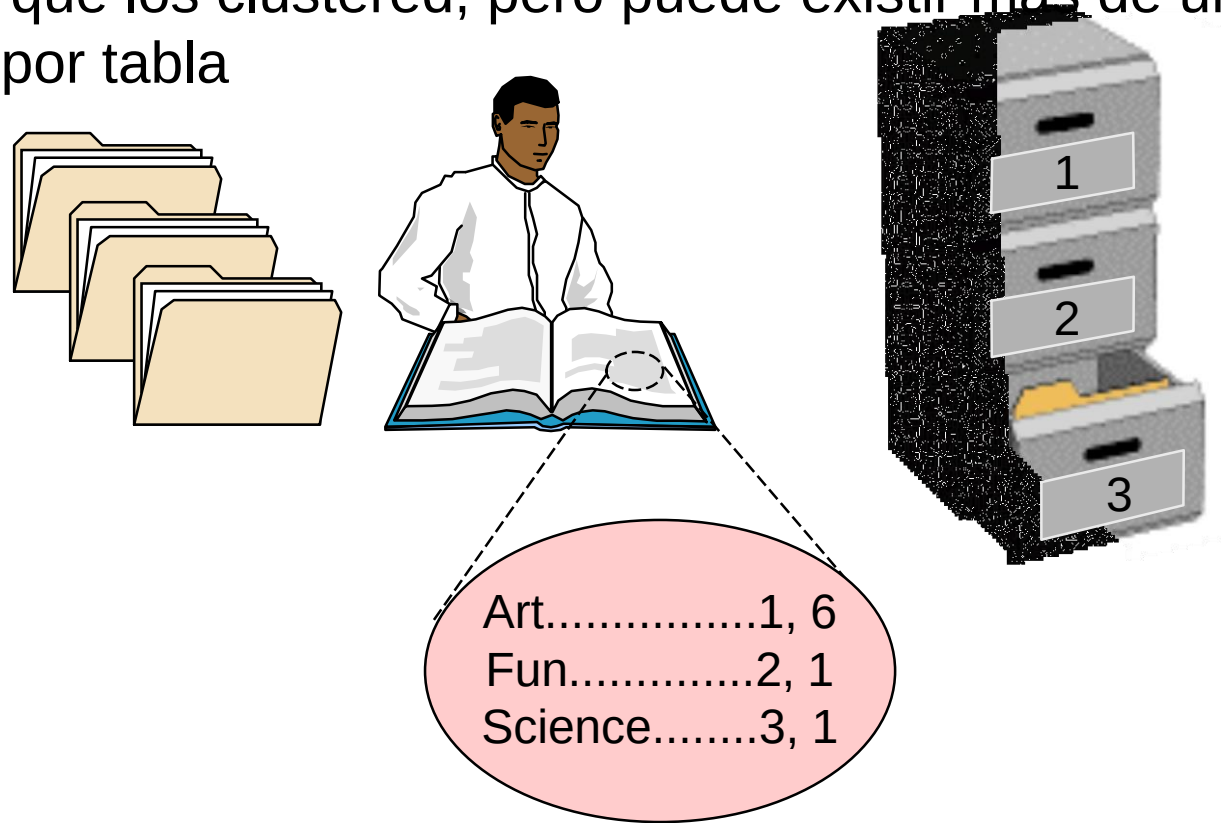
**SELECT apellido, domicilio**
**FROM persona WHERE apellido = "Green"**



# Lenguaje de definición de datos

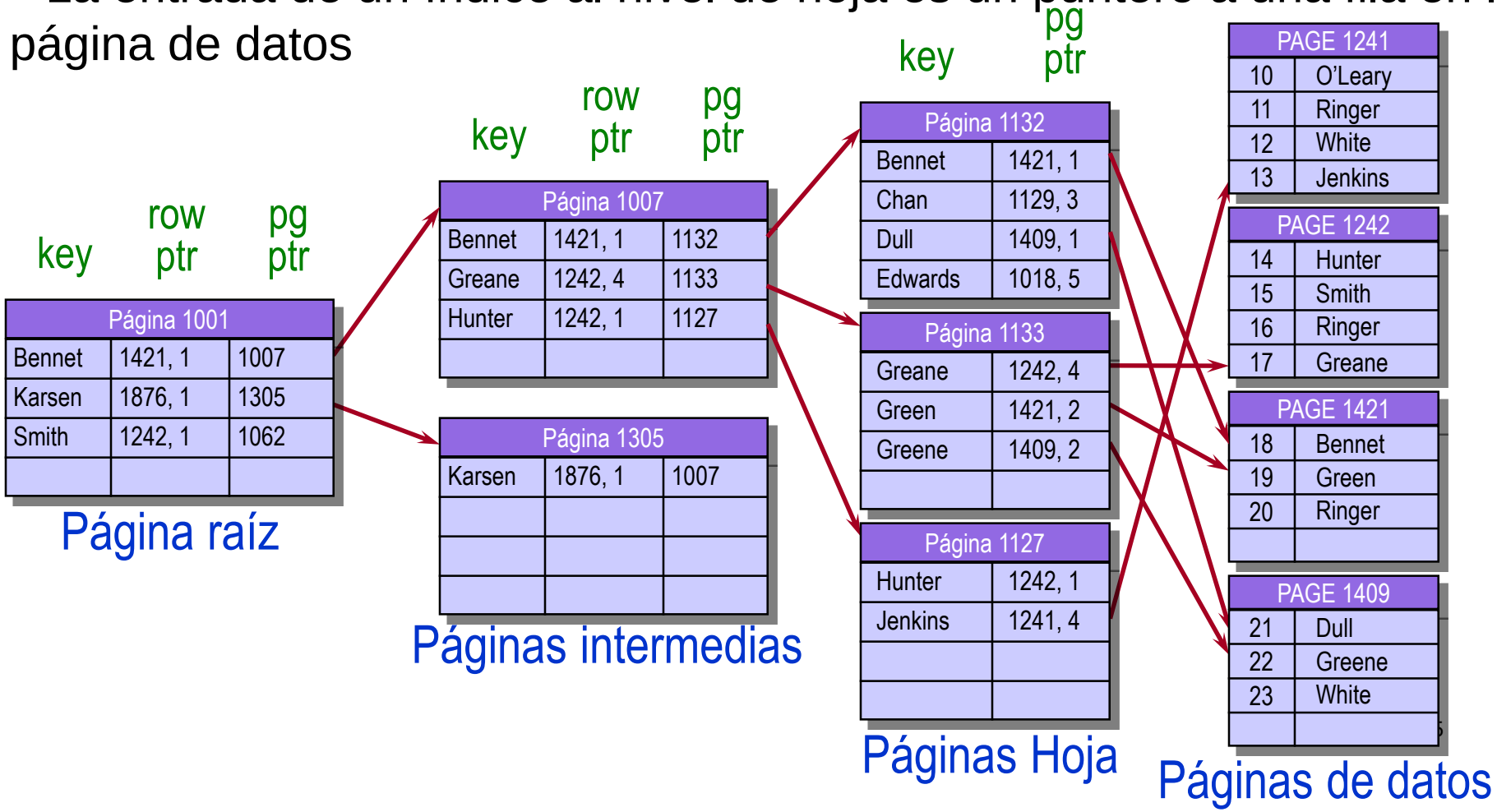
## Índices nonclustered

- La clave del índice está almacenada en orden aleatorio
- Los índices nonclustered son grandes y pueden ser menos efectivos que los clustered, pero puede existir más de uno de este tipo por tabla



# Lenguaje de definición de datos

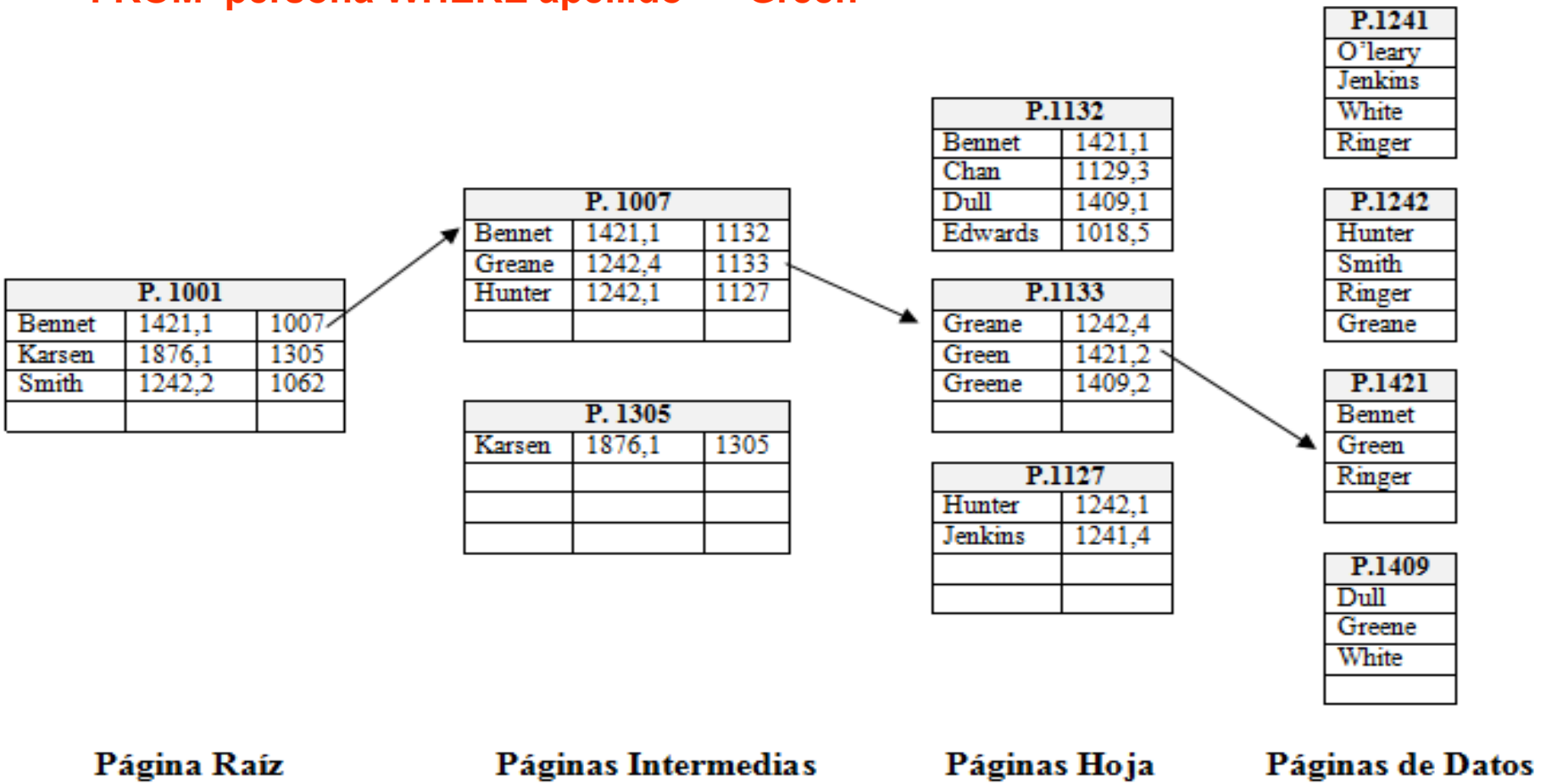
- Las páginas de nivel hoja **no** son las mismas que las de datos
- La entrada de un índice al nivel de hoja es un puntero a una fila en la página de datos

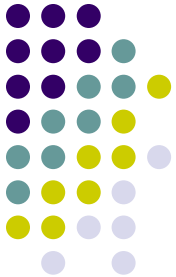


# SQL *Structured Query Language*

# Lenguaje de definición de datos

**SELECT apellido, domicilio**
**FROM persona WHERE apellido = "Green"**





# SQL *Structured Query Language*

## Lenguaje de definición de datos

### Índices

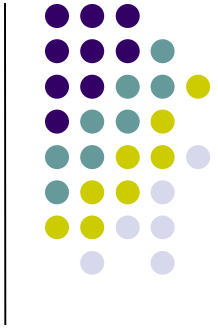
### sqlserver

```
CREATE [UNIQUE] [CLUSTERED|NONCLUSTERED] INDEX
    nombre_del_indice
    ON   [esquema.]nombre_de_la_tabla (nombre(s)_de_la(s)_columna(s));
```

### postgresql

```
CREATE [UNIQUE] INDEX nombre_del_indice
     ON  [esquema.]nombre_de_la_tabla (nombre(s)_de_la(s)_columna(s))
    [WHERE predicado];

CLUSTER [esquema.]nombre_de_la_tabla USING nombre_del_índice;
```

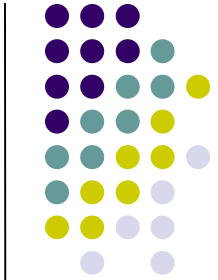
# DDL – Integridad declarativa



- La **Integridad Declarativa** es un mecanismo para implementar la integridad de datos. El server la mantiene a través del uso de cláusulas de restricciones y de valores por defecto que se incorporan en la sentencia **create table** o **alter table**, y mediante las cuales se condicionan los valores de los datos. Estas cláusulas son:
  - **Default** constraint
  - **Check** constraint
  - **Unique** constraint
  - **Primary key** constraint
  - **Reference** constraint

- Existen constraints de nivel columna y de nivel tabla.

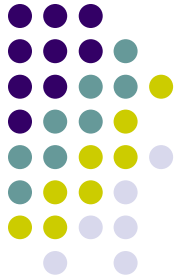
# DDL – Integridad declarativa



- La cláusula **default** pertenece al nivel columna.
- Una cláusula **default** es aplicada sobre un **insert** si no existe un valor que llene la columna.
- La cláusula **default** se utiliza para especificar el valor que ocupará la columna.
- Los **defaults** pueden ser constantes, null o *user* (char de 30 para el ultimo caso).
- Ejemplo:

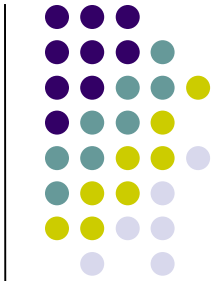
```
CREATE TABLE persona.persona
 (codigo     integer       not null,
 tipodoc     varchar(3)  DEFAULT 'DNI' not null,
 nrodoc      integer       not null,
 apenombre     varchar(60)      not null
);
```

# DDL – Integridad declarativa



- **Check** constraints:
  - Implementan integridad a los dominios.
  - Pueden ser aplicados a nivel columna o a nivel tabla.
- Son utilizados para especificar:
  - Una lista o conjunto de valores (a, b, c)
  - Un rango de valores (entre 0 y 255)
  - Un formato de edición (dos caracteres y un número: “AB3”)
  - Condiciones que debe cumplir un valor simple
- Es validado durante un **update** o **insert.**
- Especifica una “condición booleana”, restricción que el servidor de datos impone sobre todas las filas insertadas o modificadas en la tabla.

# DDL – Integridad declarativa

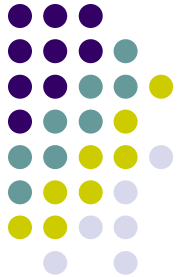


Ejemplo (check de nivel columna):

```
CREATE TABLE persona.persona
  (codigo   integer          not null,
   tipodoc  char(3)          not null
     CONSTRAINT chk_tipodoc
     CHECK (tipodoc IN ('DNI','LE','LC','PAS','OTR')),
   nrodoc   integer          not null
     CONSTRAINT chk_nrodoc CHECK (nrdoc > 1000000),
   apenom  varchar(60)       not null);
```

- Si el default viola la condición del check constraint, se rechazará cualquier inserción que use el valor default pero éste será aceptado ante la ausencia de dato para la columna en cuestión.

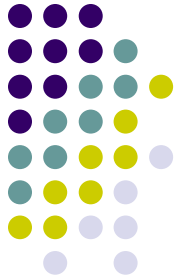
# DDL – Integridad declarativa



Ejemplo (check de nivel tabla):

```
CREATE TABLE persona.persona
  (codigo                integer          not null,
   tipodoc               char(3)          not null
     CONSTRAINT chk_tipodoc
      CHECK (tipodoc IN ('DNI','LE','LC','PAS','OTR')),
   nrodoc                integer          not null
     CONSTRAINT chk_nrodoc CHECK (nrdoc > 1000000),
   apenom                varchar(60)      not null,
   fechanacimiento       date             not null,
   fechafallecimiento    date             null,
     CONSTRAINT chk_fechanac_fechafalle
     CHECK (fechanacimiento <= fechafallecimiento)
);
```

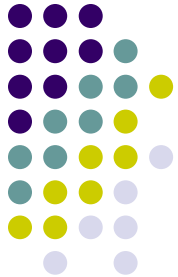
# DDL – Integridad declarativa



- Unique constraint
  - Asegura que no haya dos filas que tengan el mismo valor
  - Permite un solo valor NULL en la columna
  - Crea un **unique index nonclustered** por defecto

```
CREATE TABLE persona.medico
  (matricula     smallint     not null,
   tipodoc       char()3)     not null,
   nrodoc        integer      not null,
   CONSTRAINT unq_persona UNIQUE (tipodoc, nrodoc)
);
```

# DDL – Integridad declarativa

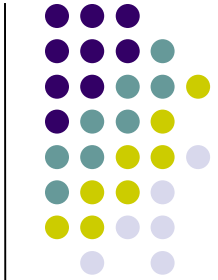


- Primary Key constraint
  - Asegura que no existan dos filas con el mismo valor
  - No admite ningún valor NULL en la columna
  - Crea un unique index ***por defecto***

Ejemplo (primary key a nivel columna):

```
CREATE TABLE persona.medico
  (matricula      smallint     not null PRIMARY KEY,
   tipodoc        char()3)     not null,
   nrodoc         integer      not null,
   CONSTRAINT unq_persona UNIQUE (tipodoc, nrodoc)
);
```

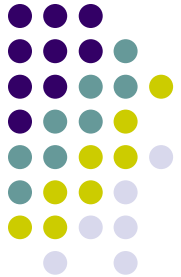
# DDL – Integridad declarativa



Ejemplo (primary key a nivel tabla):

```
CREATE TABLE persona.medico
  (matricula     smallint     not null,
   tipodoc       char()3)     not null,
   nrodoc        integer      not null,
   CONSTRAINT unq_persona UNIQUE (tipodoc, nrodoc),
   CONSTRAINT pk_medico PRIMARY KEY (matricula)
);
```

# DDL – Integridad declarativa



- Use **create table** para mantener la integridad referencial cuando una clave ajena es insertada o modificada o una clave primaria es borrada o actualizada.
- Nivel de columna:

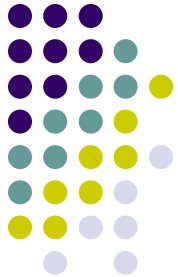
```
CREATE TABLE [esquema].nombre_tabla (
    columna tipo_dato  [CONSTRAINT nombre_constraint]
        REFERENCES [esquema.]tabla_referenciada
            [(columna_referenciada)] …
```

- Nivel de tabla:

```
[CONSTRAINT nombre_constraint]
FOREIGN KEY (nombre_columna [{, nombre_columna}..])
    REFERENCES [esquema.]tabla_referenciada
        [(columna_referenciada [{, columna_referenciada}...])]
```

- Para claves compuestas se usan constraints de nivel de tabla.

# DDL – Integridad declarativa



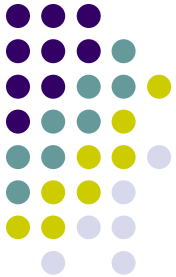
Ejemplo (integridad referencial de nivel columna):

- La tabla *especialidad* debe existir y tener un índice unique sobre *id_especialidad*
- Un valor de foreign key no puede ser insertado o modificado con un valor de *id_especialidad* inexistente en *especialidad*
- Un *id_especialidad* en *especialidad* no puede ser borrado si un valor de foreign key lo referencia

```
CREATE TABLE persona.medico
 (matricula        smallint     not null,
  tipodoc          char(3)      not null,
  nrodoc           integer      not null,
  id_especialidad  integer      not null
     REFERENCES gestion.especialidad (id_especialidad),
 CONSTRAINT unq_persona UNIQUE (tipodoc, nrodoc),
 CONSTRAINT pk_medico PRIMARY KEY (matricula)
);
```

# SQL *Structured Query Language*

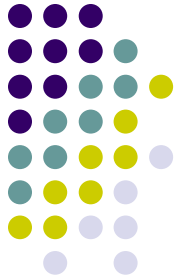
## DDL – Integridad declarativa



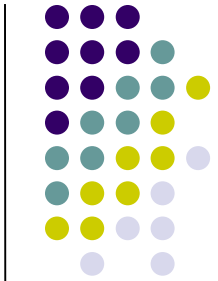
Ejemplo (integridad referencial a nivel tabla):

```
CREATE TABLE persona.medico
  (matricula          smallint     not null,
   tipodoc            char(3)      not null,
   nrodoc             integer      not null,
   id_especialidad    integer      not null,
   CONSTRAINT unq_persona UNIQUE (tipodoc, nrodoc),
   CONSTRAINT pk_medico PRIMARY KEY (matricula),
   CONSTRAINT fk_medico_persona  FOREIGN KEY (tipodoc, nrodoc)
      REFERENCES persona.persona (tipodoc, nrodoc),
   CONSTRAINT fk_medico_especialidad FOREIGN KEY (id_especialidad)
     REFERENCES gestion.especialidad (id_especialidad)
);
```

# DDL – Integridad declarativa



- **Alter Table** es usado para:
  - Agregar columnas a una tabla
  - Agregar o eliminar constraints de una tabla
  - Renombrar tablas
  - Renombrar columnas
  - Cambiar el tipo de dato
- La única manera de modificar una constraint es usando **alter table**
- Si una constraint es agregada o modificada o un default es reemplazado usando **alter table**, existiendo datos en la tabla, éstos no son afectados.



# Lenguaje de definición de datos

## Modificación de tablas

***Para añadir otra columna, la sintaxis es:***

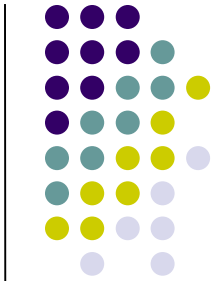
```
ALTER TABLE nombre_de_la_tabla
    ADD COLUMN nombre_de_la_columna  tipo_de_datos;
```

***Para cambiar el ancho de una columna ya existente, la sintaxis es:***

```
ALTER TABLE  nombre_de_la_tabla
    ALTER COLUMN nombre_de_la_columna
    TYPE tipo_de_datos    nueva_anchura;
```

***Para modificar el nombre de una columna:***

```
ALTER TABLE nombre_tabla
    RENAME nombre_viejo TO nombre_nuevo;
```



# Lenguaje de definición de datos

## Modificación de tablas

***Para eliminar una columna:***

```
ALTER TABLE  nombre_de_la_tabla
      DROP COLUMN nombre_de_la_columna;
```

***Para eliminar o agregar restricciones:***

```
ALTER TABLE nombre_de_la_tabla
      ADD CONSTRAINT ... (igual definición que en la tabla);

ALTER TABLE nombre_de_la_tabla
      DROP CONSTRAINT nombre_constraint;
```

# Lenguaje de definición de datos

## Eliminación

***La supresión de bases de datos:***
***DROP DATABASE nombre_de_la_base;***

***La supresión de tablas:***
***DROP TABLE  nombre_de_la_tabla;***

***La supresión de vistas:***
***DROP VIEW nombre_de_la_vista;***

***La supresión de índices:***
***DROP INDEX  nombre_del_índice ON nombre_de_la_tabla;***

# SQL *Structured Query Language*

# Lenguaje de control de datos

## GRANT - REVOKE

| Permiso | Objeto |
|---|---|
| SELECT | Tabla, Vista o Columna |
| UPDATE | Tabla, Vista o Columna |
| INSERT | Tabla, Vista |
| DELETE | Tabla, Vista |
| REFERENCES | Tabla, Columna |
| EXECUTE | Stored procedure |

**GRANT privilegio_nivel_base TO usuario [WITH GRANT OPTION];**

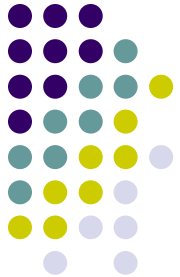
Otorga permisos en el ámbito de base de datos completa.

**GRANT privilegio_nivel_tabla ON nombre_de_la_tabla TO usuario [WITH GRANT OPTION];**

Otorga permisos a nivel de tabla.

# SQL *Structured Query Language*

# Lenguaje de control de datos



## GRANT - REVOKE

**GRANT ALL PRIVILEGES TO user1 WITH GRANT OPTION;**

**Todos los privilegios al USER1 y puede a su vez otorgarlos a otros.**

**GRANT ALL PRIVILEGES ON tabla1, tabla2 TO user2;**

**Todos los privilegios al USER2 sobre las tablas “tabla1” y “tabla2”.**

**GRANT ALL PRIVILEGES**
**ON tabla1(col1), tabla2(col1) TO user2, user3, user4;**

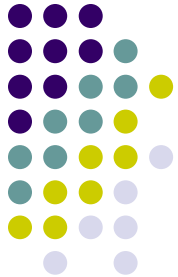
**Todos los privilegios al USER2, USER3 y USER4 sobre las columnas col1 de la tabla1 y col1 de la tabla2.**

**GRANT SELECT ON tabla1 TO PUBLIC;**

**Privilegio de consulta a *cualquier usuario* sobre la “tabla1”.**

# SQL *Structured Query Language*

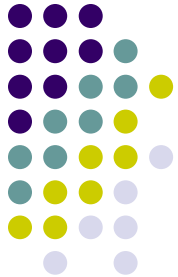
# Lenguaje de control de datos



## GRANT - REVOKE

```
GRANT {ALL [PRIVILEGES] | lista_de_permisos }
      ON
      {
        nombre_de_tabla [(lista_de_columnas)]
         | nombre_de_vista [(lista_de_columnas)]
         |   nombre_procedimiento_almacenado
       }
      TO { PUBLIC | lista_de_usuarios  |  nombre_del_rol }
      [WITH GRANT OPTION]
```

# Lenguaje de control de datos



## GRANT - REVOKE

```
REVOKE
    [GRANT OPTION FOR] {ALL [PRIVILEGES] | lista_de_permisos }
    ON
       {
        nombre_de_tabla [(lista_de_columnas)]
        |   nombre_de_vista [(lista_de_columnas)]
        |   nombre_procedimiento_almacenado
       }
    FROM  { PUBLIC | lista_de_usuarios  |  nombre_del_rol }
```

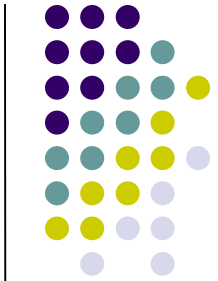
# Lenguaje de control de datos

## Transacciones

**BEGIN TRANSACTION;**

**COMMIT TRANSACTION;**

**ROLLBACK TRANSACTION;**

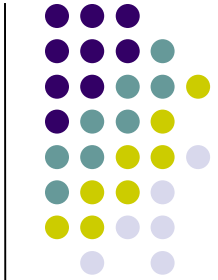


# **ACID** (**A**tomicity, **C**onsistency, **I**solation and **D**urability)

Conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción. Es un acrónimo **A**tomicidad, **C**onsistencia, a**I**slamiento y **D**urabilidad.

Atomicidad: es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.

Consistencia: Integridad. Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido.



# **ACID** (**A**tomicity, **C**onsistency, **I**solation and **D**urability)

Aislamiento: es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.

Durabilidad: es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

Cumpliendo estos 4 requisitos un sistema gestor de bases de datos puede ser considerado ACID Compliant.