



Universidad del
Rosario

Educación Continua
y Consultoría

Diplomado en Ciencia de Datos



#URSolucionesInnovadoras

#URConsultoría



Universidad del
Rosario

Educación Continua
y Consultoría

BASES DE DATOS 4

Modelado de bases de datos



Contenido

- Instalación de PostgreSQL
- Lenguaje de Definición de datos (DDL)
- Tipos de datos
- Restricciones de integridad
- Ejercicio práctico: Car Portal Database
- Características de un buen diseño relacional





Contenido

- **Instalación de PostgreSQL**
- Lenguaje de Definición de datos (DDL)
- Tipos de datos
- Restricciones de integridad
- Ejercicio práctico: Car Portal Database
- Características de un buen diseño relacional





Universidad del
Rosario

Educación Continua
y Consultoría

Instalación

Creando un servidor de base de datos



PostgreSQL: The World's Most Advanced Open Source
Relational Database

Download →

New to PostgreSQL?

<https://www.postgresql.org/>

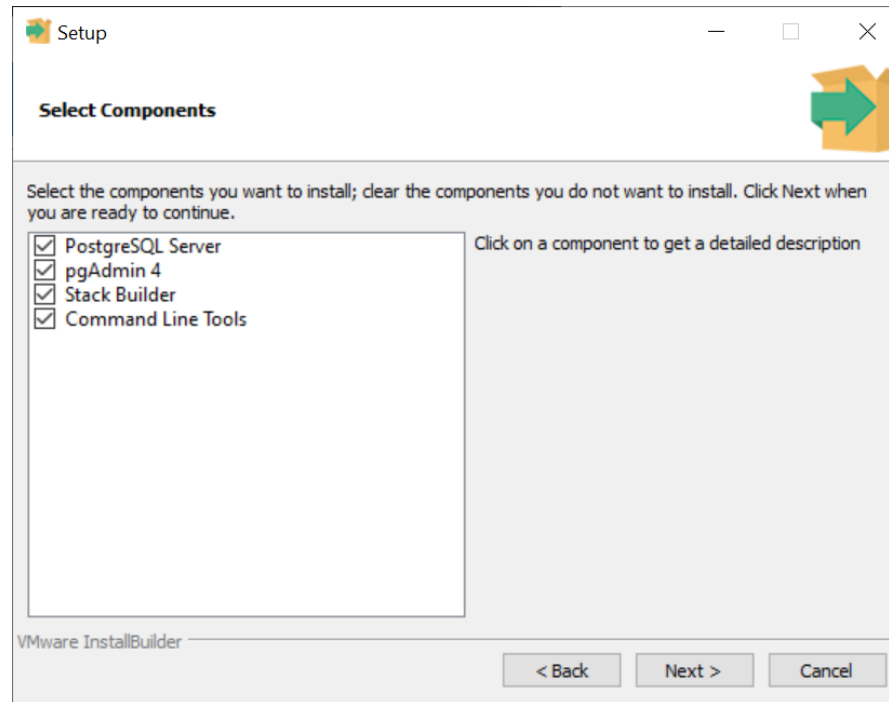


Universidad del
Rosario

Educación Continua
y Consultoría

Instalación

Creando un servidor de base de datos



<https://www.postgresql.org/>





Contenido

- Instalación de PostgreSQL
- **Lenguaje de Definición de datos (DDL)**
- Tipos de datos
- Restricciones de integridad
- Ejercicio práctico: Car Portal Database
- Características de un buen diseño relacional



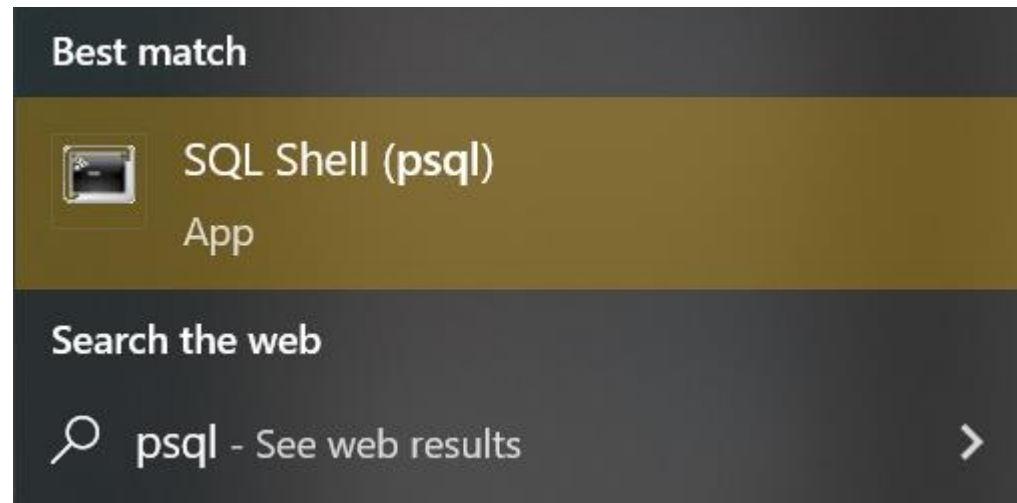


Universidad del
Rosario

Educación Continua
y Consultoría

Lenguaje de Definición de Datos (DDL)

Conexión a psql





Lenguaje de Definición de Datos (DDL)

Creación de base de datos

- Activar el modo expandido con `\x`
- Listar todas las bases de datos con `\l`
- Conectarse a una base de datos particular con `\c`
<nombrebasededatos>

- Crear una base de datos con

```
CREATE DATABASE databasename
```





Lenguaje de Definición de Datos (DDL)

Borrar tablas y bases de datos

- Borrar tablas

```
drop table dummytable;
```

- Borrar database

```
drop database dummydb;
```





Lenguaje de Definición de Datos (DDL)

Crear tablas

Síntaxis completa en <http://www.postgresql.org/docs/current/static/sql-createtable.html>

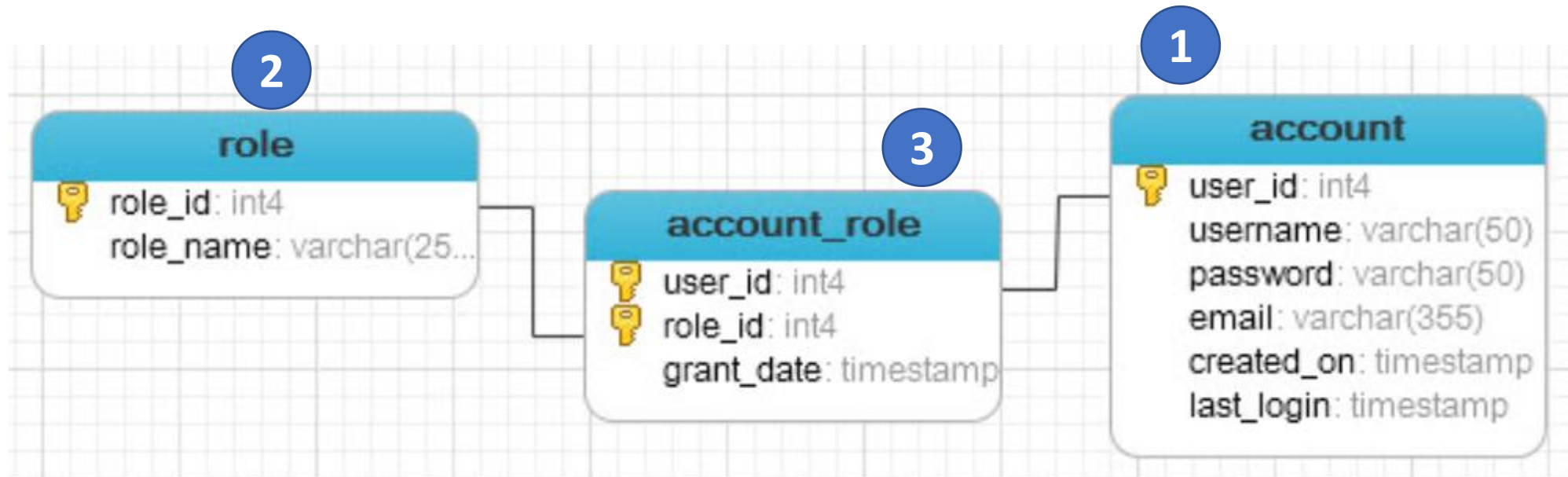
El comando SQL **CREATE TABLE** normalmente requiere la siguiente entrada:

- Nombre de tabla de la tabla creada.
- El tipo de tabla.
- Las columnas de la tabla, incluido el tipo de datos, los valores predeterminados y la restricción.



Lenguaje de Definición de Datos (DDL)

Esquema de ejemplo






Lenguaje de Definición de Datos (DDL)

Creación de tablas

Declaración de una tabla con **CREATE TABLE**

1

account	
	user_id: int4
	username: varchar(50)
	password: varchar(50)
	email: varchar(355)
	created_on: timestamp
	last_login: timestamp

```
CREATE TABLE account(  
    user_id serial PRIMARY KEY,  
    username VARCHAR (50) UNIQUE NOT NULL,  
    password VARCHAR (50) NOT NULL,  
    email VARCHAR (355) UNIQUE NOT NULL,  
    created_on TIMESTAMP NOT NULL,  
    last_login TIMESTAMP  
);
```



Lenguaje de Definición de Datos (DDL)

Eliminación de tablas

Es posible eliminar toda la tabla, incluidas todas las tuplas actuales.

- La relación *movies* ya no hará parte del esquema de la base de datos.
- No se podrá acceder a las tuplas de *movies*.

```
DROP TABLE movies;
```



Lenguaje de Definición de Datos (DDL)

Actualización de datos

- Sirven para modificar el esquema de una tabla existente.
- Estas modificaciones se realizan mediante una declaración que comienza con las palabras clave **ALTER TABLE** y el nombre de la relación.
- Tiene dos operaciones posibles:
 - **ADD** seguido por el nombre del atributo y su tipo de dato.

```
ALTER TABLE account ADD score INT;
```

- **DROP** seguido por el nombre del atributo.

```
ALTER TABLE account DROP password;
```




Lenguaje de Definición de Datos (DDL)

Cambiar el tipo de dato de una columna



```
ALTER TABLE account  
ALTER COLUMN score TYPE VARCHAR(30);
```

En caso de que falle la conversión implícita, es necesario usar la cláusula USING, la cual permite especificar una expresión que convierte los valores antiguos a los nuevos

```
ALTER TABLE account  
ALTER COLUMN score TYPE INT;
```

```
ALTER TABLE account  
ALTER COLUMN score TYPE INT  
USING score::INTEGER;
```



Lenguaje de Definición de Datos (DDL)

Tablas de ejemplo



2



```
CREATE TABLE role(  
    role_id serial PRIMARY KEY,  
    role_name VARCHAR (255) UNIQUE NOT NULL  
);
```



Lenguaje de Definición de Datos (DDL)

Llaves foráneas y tablas intermedias

```
CREATE TABLE account_role
(
  user_id integer NOT NULL,
  role_id integer NOT NULL,
  grant_date timestamp without time zone,
  PRIMARY KEY (user_id, role_id),
  CONSTRAINT account_role_role_id_fkey FOREIGN KEY (role_id)
  REFERENCES role (role_id) MATCH SIMPLE
  ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT account_role_user_id_fkey FOREIGN KEY (user_id)
  REFERENCES account (user_id) MATCH SIMPLE
  ON UPDATE NO ACTION ON DELETE NO ACTION
);
```





Contenido

- Instalación de PostgreSQL
- Lenguaje de Definición de datos (DDL)
- **Tipos de datos**
- Restricciones de integridad
- Ejercicio práctico: Car Portal Database
- Características de un buen diseño relacional





Tipos de datos

Tipos de datos existentes

- Numéricos
- Textos
- Temporales
- Arreglos
- Rango
- JSON
- XML
- Full text search
- Custom y compuestos





Tipos de datos

Datos numéricos

- Similar al lenguaje C, el resultado de una expresión entera también es un número entero.
- Los resultados de las operaciones matemáticas $3/2$ y $1/3$ son 1 y 0, respectivamente. Por lo tanto, la parte fraccionaria siempre se trunca.



Name	Comments	Size	Range
small-int	SQL equivalent: Int2	2 bytes	-32,768 to +32,767.
integer	SQL equivalent: Int4 Integer is an alias for INT.	4 bytes	-2,147,483,648 to +2,147,483,647.
bigint	SQL equivalent: Int8 8 bytes	8 bytes	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.
numeric or decimal	No difference in PostgreSQL	Variable	Up to 131,072 digits before the decimal point; up to 16,383 digits after the decimal point.
real	Special values: Infinity, -Infinity, NaN	4 bytes	Platform-dependent, at least six-digit precision. Often, the range is 1E-37 to 1E+37.
double precision	Special values: Infinity, -Infinity, NaN	8 bytes	Platform dependent, at least 15-digit precision. Often, the range is 1E-307 to 1E+308.



Datos numéricos

Ejemplo de operación de números

Tres formas de crear números

- Numeric(precision, scale)
- Numeric(precision)
- Numeric



Escala = 3

1234.567

Precisión = 7



Datos numéricos

Ejemplo de operación de números



```
SELECT  
CAST(5.9 AS INT) AS "CAST (5.9 AS INT)",  
CAST(5.1 AS INT) AS "CAST(5.1 AS INT)",  
CAST(-23.5 AS INT) AS "CAST(-23.5 AS INT)"  
,  
5.5::INT AS "5.5::INT";
```

```
CREATE TABLE customer (  
  customer_id NUMERIC  
);
```



Universidad del
Rosario

Educación Continua
y Consultoría

Datos numéricos

Ejemplo de operación de números



```
CREATE TABLE customer (  
  customer_id NUMERIC,  
  customer_name VARCHAR(50)  
);
```

¿Qué opinas sobre los tipos de datos en esta tabla?



Tipos de datos

Datos numéricos



- Autoincrementales: serial y bigserial

```
CREATE SEQUENCE s START 1;  
CREATE TABLE stuff(id bigint DEFAULT  
nextval('s') PRIMARY KEY, name text);
```

- Atributo de tipo serial

```
CREATE TABLE table_name(  
    id SERIAL  
);
```

- Función generadora de series

```
SELECT x FROM generate_series(1,51,13) As  
x;
```



Tipos de datos

Datos numéricos

Name	Storage Size	Range
SMALLSERIAL	2 bytes	1 to 32,767
SERIAL	4 bytes	1 to 2,147,483,647
BIGSERIAL	8 bytes	1 to 9,223,372,036,854,775,807



Tipos de datos

Datos de texto

Tres tipos principales

- char
- varchar
- Text

Algunas funciones de texto

```
SELECT  
  lpad('ab', 4, '0') As ab_lpad,  
  rpad('ab', 4, '0') As ab_rpad,  
  lpad('abcde', 4, '0') As ab_lpad_trunc;
```

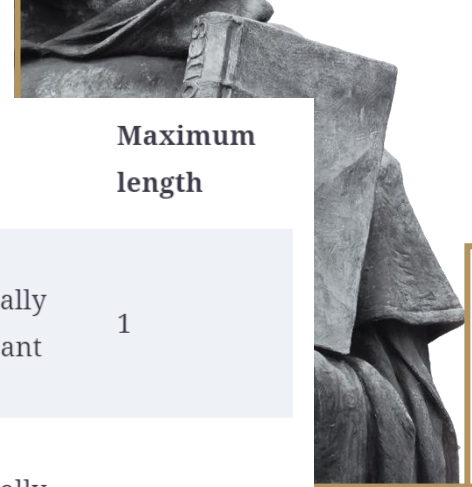




Tipos de datos

Datos de texto

Name	Comments	Trailing spaces	Maximum length
char	Equivalent to char(1), it must be quoted as shown in the name.	Semantically insignificant	1
name	Equivalent to varchar(64). Used by Postgres for object names.	Semantically significant	64
char(n)	Alias: character(n). Fixed-length character where the length is n. Internally called blank padded character (bpchar) .	Semantically insignificant	1 to 10485760
varchar(n)	Alias: character varying(n). Variable-length character where the maximum length is n.	Semantically significant	1 to 10485760
text	Variable-length character.	Semantically significant	Unlimited





Tipos de datos

Un experimento con texto

Creación de tablas

```
CREATE TABLE char_size_test (  
  size CHAR(10)  
);  
CREATE TABLE varchar_size_test(  
  size varchar(10)  
);
```

Generación de datos aleatorios

```
WITH test_data AS (  
  SELECT substring(md5(random()::text), 1, 5) FROM generate_series (1, 1000000)  
) , char_data_insert AS (  
  INSERT INTO char_size_test SELECT * FROM test_data  
) INSERT INTO varchar_size_test SELECT * FROM test_data;
```

Comprobación de tamaño

```
\dt+ varchar_size_test
```

```
\dt+ char_size_test
```





Tipos de datos

Expresiones regulares



- Patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto.
- Las expresiones regulares proporcionan una manera muy flexible de buscar o reconocer cadenas de texto.

<https://regexr.com/>

```
SELECT regexp_replace(  
  '6197306254',  
  '([0-9]{3})([0-9]{3})([0-9]{4})',  
  E'\\(\\1\\) \\2-\\3'  
) As x;
```



Tipos de datos

Tipos de dato y fecha

- PostgreSQL almacena la marca de tiempo con y sin la zona horaria en el formato de hora universal coordinada (UTC),
- Solo se almacena la hora sin la zona horaria. Esto explica el tamaño de almacenamiento idéntico tanto para la marca de tiempo con la zona horaria como para la marca de tiempo sin zona horaria.

Name	Size in bytes	Description	Low value	High value
Timestamp without time zone	8	Date and time without time zone, equivalent to timestamp	4713 BC	294276 AD
Timestamp with time zone	8	Date and time with time zone, equivalent to timestampz	4713 BC	294276 AD
Date	4	Date only	4713 BC	294276 AD
Time without time zone	8	Time of day	00:00:00	24:00:00
Time with time zone	12	Time of day with time zone	00:00:00+1459	24:00:00-1459
Interval	16	Time interval	-178,000,000 years	+178,000,000 years



Universidad del
Rosario

Educación Continua
y Consultoría

Tipos de datos

Tipos de dato y fecha

Un experimento de cambio de zona horaria

```
SELECT now();
```

```
SET timezone TO 'Asia/jerusalem';
```

```
SHOW timezone;
```

Verificar las diferentes zonas en <https://www.postgresql.org/docs/8.1/datetime-keywords.html>





Contenido

- Instalación de PostgreSQL
- Lenguaje de Definición de datos (DDL)
- Tipos de datos
- **Restricciones de integridad**
- Ejercicio práctico: Car Portal Database
- Características de un buen diseño relacional





Restricciones de integridad

¿Qué son?

Reglas que se aseguran de que la base de datos no pierda consistencia cuando se hacen cambios en los datos

Ejemplos:

- El nombre de un instructor no puede ser nulo
- Dos instructores no pueden tener el mismo ID
- Cada departamento que aparezca en curso debe existir en la tabla departamento
- El presupuesto de cada departamento debe ser mayor a cero





Restricciones de integridad

Ejemplos de definición de restricciones

```
CREATE TABLE habitante (  
    id_habitante serial NOT NULL,  
    nombre varchar(60) NOT NULL,  
    tipo_documento char(2) NOT NULL,  
    num_documento varchar(12) NOT NULL,  
    email varchar(60) NOT NULL,  
    id_habitante_host integer,  
    CONSTRAINT habitante_pk PRIMARY KEY (id_habitante),  
    CONSTRAINT unique_email UNIQUE (email)  
);
```

```
ALTER TABLE distributors  
ADD PRIMARY KEY (dist_id); -- PK
```

```
ALTER TABLE distributors  
ADD CONSTRAINT distfk FOREIGN KEY (address) REFERENCES addresses (address); -- FK
```

```
UPDATE habitante ADD unique_email UNIQUE (email); --UNIQUE
```

```
ALTER TABLE habitante  
ADD CONSTRAINT check_zip CHECK (char_length(zipcode) = 5) -- CHECK;
```

```
ALTER TABLE habitante  
ALTER COLUMN id_habitante_host SET NOT NULL; -- NN
```





Universidad del
Rosario

Educación Continua
y Consultoría

Restricciones de integridad

Tipos de restricciones

- CHECK
- NOT-NULL
- UNIQUE
- PRIMARY KEYS
- FOREIGN KEYS





Restricciones de integridad

Restricciones en una tabla

- NOT NULL
- UNIQUE
- CHECK

```
CREATE TABLE habitante (  
    id_habitante SERIAL NOT NULL,  
    nombre VARCHAR(60) NOT NULL,  
    tipo_documento CHAR(2) NOT NULL,  
    num_documento VARCHAR(12) NOT NULL,  
    email VARCHAR(60) NOT NULL,  
    id_habitante_host INTEGER,  
    CONSTRAINT habitante_pk PRIMARY KEY (id_habitante),  
    CONSTRAINT unique_email UNIQUE (email),  
    CHECK ( LENGTH(num_documento) > 5 )  
);
```



Restricciones de integridad

CHECK constraint

El valor debe satisfacer una expresión booleana

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric CHECK (price > 0)  
);
```

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric CONSTRAINT positive_price CHECK (price > 0)  
);
```





Restricciones de integridad

CHECK constraint

Se puede hacer referencia a varias columnas

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric CHECK (price > 0),  
    discounted_price numeric CHECK  
(discounted_price > 0),  
    CHECK (price > discounted_price)  
);
```





Restricciones de integridad

CHECK constraint

Varias sintaxis

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric,  
    CHECK (price > 0),  
    discounted_price numeric,  
    CHECK (discounted_price > 0),  
    CHECK (price > discounted_price)  
);
```

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric CHECK (price > 0),  
    discounted_price numeric,  
    CHECK (discounted_price > 0 AND price > discounted_price)  
);
```



Restricciones de integridad

CHECK constraint



Varias sintaxis

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric,  
    CHECK (price > 0),  
    discounted_price numeric,  
    CHECK (discounted_price > 0),  
    CONSTRAINT valid_discount CHECK (price > discounted_price)  
);
```



Restricciones de integridad

NOT-NULL constraint

Especifica que el valor debe existir

```
CREATE TABLE products (  
    product_no integer NOT NULL,  
    name text NOT NULL,  
    price numeric  
);
```





Restricciones de integridad

NOT-NULL constraint

Una columna puede tener más de una restricción

```
CREATE TABLE products (  
    product_no integer NOT NULL,  
    name text NOT NULL,  
    price numeric NOT NULL CHECK (price > 0)  
);
```





Restricciones de integridad

UNIQUE constraint

El valor que se ingresa es único

```
CREATE TABLE products (  
    product_no integer UNIQUE,  
    name text,  
    price numeric  
);
```





Restricciones de integridad

UNIQUE constraint

Otras sintaxis

```
CREATE TABLE products (  
    product_no integer CONSTRAINT must_be_different UNIQUE,  
    name text,  
    price numeric  
);
```

```
CREATE TABLE example (  
    a integer,  
    b integer,  
    c integer,  
    UNIQUE (a, c)  
);
```

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric,  
    UNIQUE (product_no)  
);
```



Restricciones de integridad

Restricciones de llave primaria y foránea (varias tablas)

```
CREATE TABLE alquiler (  
  id_alquiler SERIAL NOT NULL,  
  inicio_alquiler DATE NOT NULL,  
  fin_alquiler DATE,  
  canon_alquiler NUMERIC NOT NULL,  
  id_propiedad INTEGER NOT NULL,  
  id_habitante INTEGER NOT NULL,  
  CONSTRAINT alquiler_pk PRIMARY KEY (id_alquiler),  
  CONSTRAINT chk_canon_alquiler CHECK (canon_alquiler >= 0)  
);
```



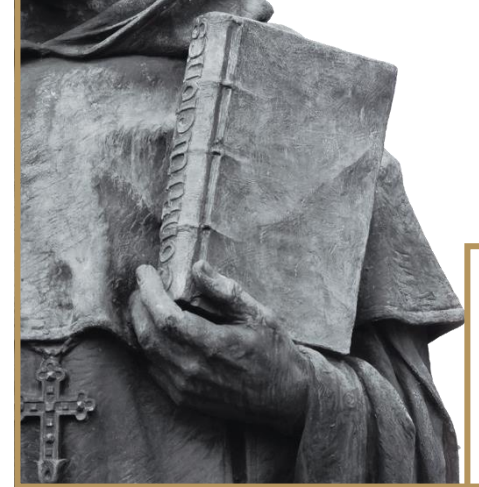
Restricciones de integridad

PRIMARY KEY constraint

Identificador único no nulo en la tabla

```
CREATE TABLE products (  
    product_no integer PRIMARY KEY,  
    name text,  
    price numeric  
);
```

```
CREATE TABLE example (  
    a integer,  
    b integer,  
    c integer,  
    PRIMARY KEY (a, c)  
);
```





Restricciones de integridad

FOREIGN KEY constraint

Los valores de la columna deben hacer match con los identificadores de otra tabla

```
CREATE TABLE products (  
    product_no integer PRIMARY KEY,  
    name text,  
    price numeric  
);
```

```
CREATE TABLE orders (  
    order_id integer PRIMARY KEY,  
    product_no integer REFERENCES products (product_no),  
    quantity integer  
);
```



Restricciones de integridad

FOREIGN KEY constraint

Se puede abreviar el comando

```
CREATE TABLE orders (  
    order_id integer PRIMARY KEY,  
    product_no integer REFERENCES products,  
    quantity integer  
);
```





Restricciones de integridad

FOREIGN KEY constraint

Múltiples columnas en la llave

```
CREATE TABLE t1 (  
  a integer PRIMARY KEY,  
  b integer,  
  c integer,  
  FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)  
);
```





Restricciones de integridad

FOREIGN KEY constraint

RESTRICT previene el borrado

NO ACTION lanza un error

CASCADE borra hasta las referenciadas

```
CREATE TABLE products (  
    product_no integer PRIMARY KEY,  
    name text,  
    price numeric  
);
```

```
CREATE TABLE orders (  
    order_id integer PRIMARY KEY,  
    shipping_address text,  
    ...  
);
```

```
CREATE TABLE order_items (  
    product_no integer REFERENCES products ON DELETE RESTRICT,  
    order_id integer REFERENCES orders ON DELETE CASCADE,  
    quantity integer,  
    PRIMARY KEY (product_no, order_id)  
);
```



Restricciones de integridad

FOREIGN KEY constraint

Hay ON DELETE y ON UPDATE

```
CREATE TABLE order_items (  
  product_no integer REFERENCES products ON DELETE RESTRICT,  
  order_id integer REFERENCES orders ON DELETE CASCADE,  
  quantity integer,  
  PRIMARY KEY (product_no, order_id)  
);
```





Restricciones de integridad

FOREIGN KEY constraint

Cuando se borran registros se puede poner SET NULL o SET DEFAULT para establecer los valores que quedan

```
CREATE TABLE order_items (  
    product_no integer REFERENCES products ON DELETE RESTRICT,  
    order_id integer REFERENCES orders ON DELETE SET NULL,  
    quantity integer,  
    PRIMARY KEY (product_no, order_id)  
);
```





Contenido

- Instalación de PostgreSQL
- Lenguaje de Definición de datos (DDL)
- Tipos de datos
- Restricciones de integridad
- **Ejercicio práctico: Car Portal Database**
- Características de un buen diseño relacional





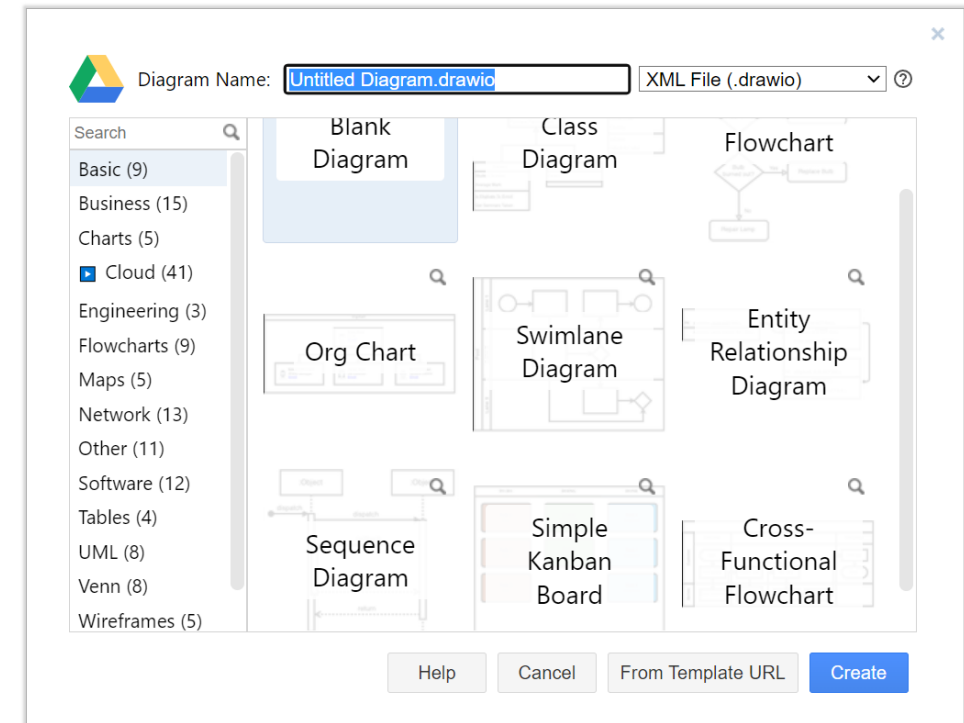
Ejercicio práctico: Car Portal Database

Generación de modelo ER

Con el fin de explicar los conceptos básicos del modelo ER, se modelará un portal web en línea para comprar y vender automóviles.

La idea es generar el modelo ER que soporta los requerimientos de negocio

<https://app.diagrams.net/>





Ejercicio práctico: Car Portal Database

Requerimientos de la aplicación



1. El portal brinda la posibilidad de registrar usuarios en línea y brinda diferentes servicios para los usuarios según sus categorías.
2. Los usuarios pueden ser vendedores o usuarios normales.
3. Los vendedores pueden crear anuncios de autos nuevos; otros usuarios pueden explorar y buscar autos.
4. Todos los usuarios deben proporcionar su nombre completo y una dirección de correo electrónico válida durante el registro. La dirección de correo electrónico se utilizará para iniciar sesión.
5. El vendedor también debe proporcionar una dirección.
6. El usuario puede calificar el anuncio y la calidad del servicio del vendedor.
7. El historial de búsqueda de todos los usuarios debe mantenerse para su uso posterior.
8. Los vendedores tienen rangos y esto afecta la búsqueda de anuncios; el rango está determinado por el número de anuncios publicados y el rango del usuario.
9. El anuncio de automóvil tiene una fecha y el automóvil puede tener muchos atributos, como color, número de puertas, número de propietarios anteriores, número de registro, imágenes, etc.



Ejercicio práctico: Car Portal Database

Sentencias DDL de Car Portal Database

https://drive.google.com/file/d/14fEy15y8ALHv3XoSGQGaRofmm8B_vjm/view?usp=sharing

Falta una tabla que debe ser creada por cada uno y debe almacenar la siguiente información:

- car_id = ID del automóvil
- manufacturer = Nombre del Fabricante
- model = nombre del modelo
- displ = desplazamiento del cilindro, en litros
- year = año de manufactura
- cyl = Número de cilindros
- trans = tipo de transmisión
- drv = el tipo de tren de transmisión, donde f = tracción delantera, r = tracción trasera, 4 = 4wd
- cty = millas de la ciudad por galón
- hwy = millas de carretera por galón
- fl = tipo de combustible ["corriente", "extra", "diesel", "eléctrico"]
- class = Tipo de vehículo ["compact", "midsize", "suv", "2seater", "minivan", "pickup", "subcompact"]



Ejercicio práctico: Car Portal Database

Sentencias DDL de Car Portal Database

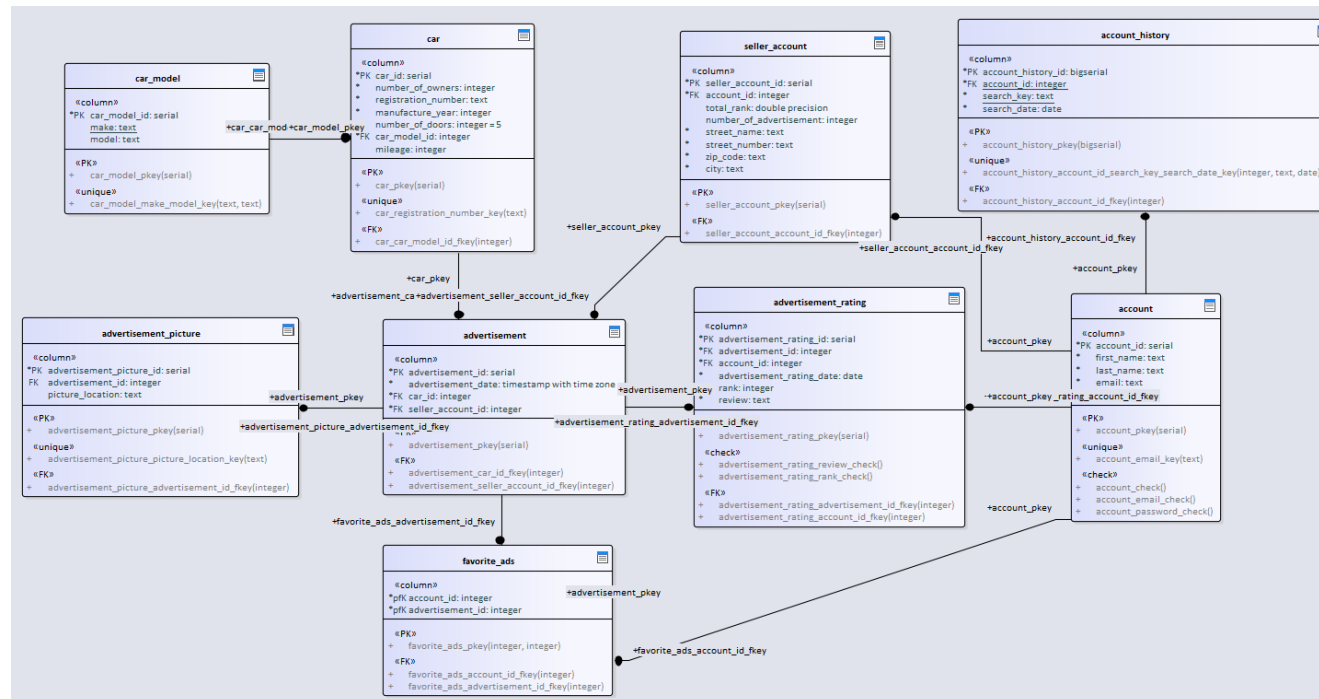


manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
chevrolet	corvette	5.7	1999	8	auto(l4)	r	15	23	p	2seater
chevrolet	corvette	6.2	2008	8	manual(m6)	r	16	26	p	2seater
chevrolet	corvette	6.2	2008	8	auto(s6)	r	15	25	p	2seater
chevrolet	corvette	7.0	2008	8	manual(m6)	r	15	24	p	2seater
chevrolet	k1500 tahoe 4wd	5.3	2008	8	auto(l4)	4	14	19	r	suv
chevrolet	k1500 tahoe 4wd	5.3	2008	8	auto(l4)	4	11	14	e	suv
chevrolet	k1500 tahoe 4wd	5.7	1999	8	auto(l4)	4	11	15	r	suv
chevrolet	k1500 tahoe 4wd	6.5	1999	8	auto(l4)	4	14	17	d	suv
chevrolet	malibu	2.4	1999	4	auto(l4)	f	19	27	r	midsize
chevrolet	malibu	2.4	2008	4	auto(l4)	f	22	30	r	midsize



Ejercicio práctico: Car Portal Database

El modelo





Características de un buen diseño relacional

¿Qué es un buen diseño?

Es un diseño que se encuentra en su *forma normal* apropiada

Ventajas

- Poca redundancia
- Información fácil de consultar

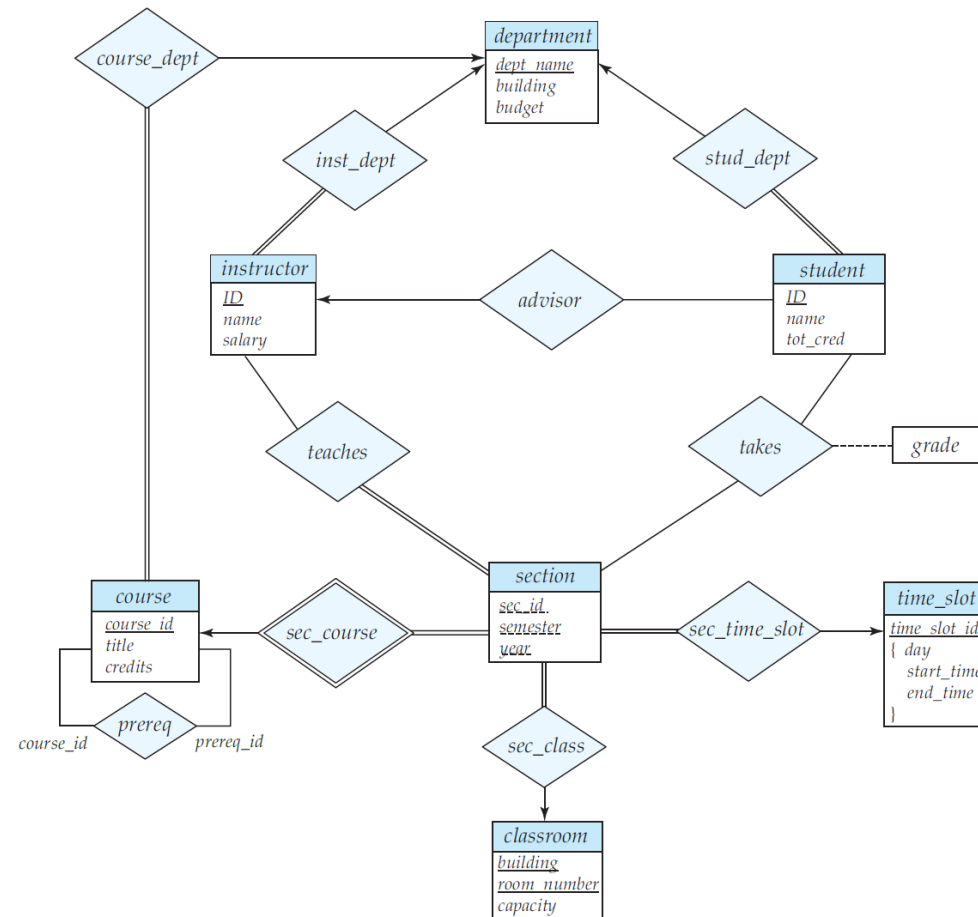
Información necesaria para el modelo

- Diagrama E-R
- Información adicional del negocio según se requiera



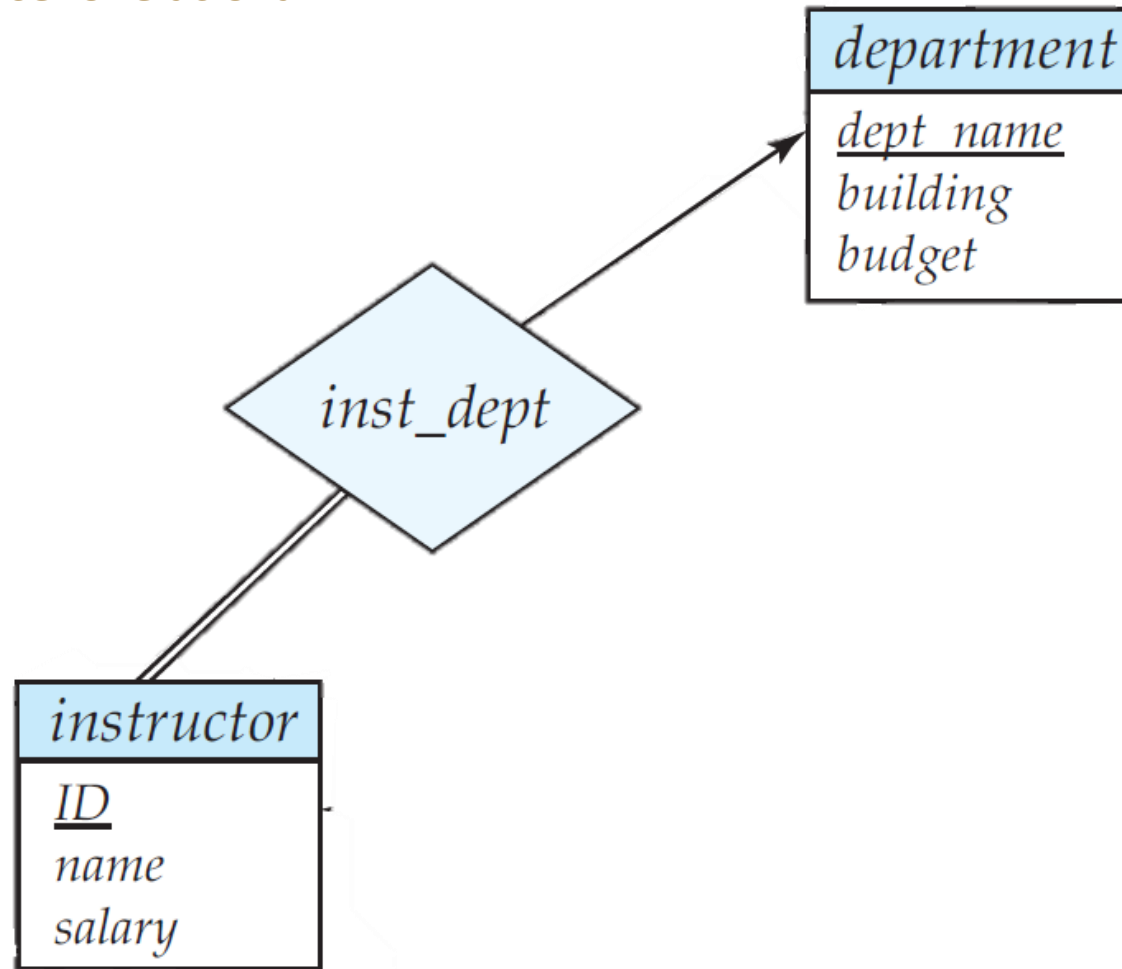


Características de un buen diseño relacional





Características de un buen diseño relacional





Características de un buen diseño relacional

Alternativas de diseño

Esquemas grandes

■ Ventajas:

- Tenerlos unidos ahorra operaciones de JOIN

■ Desventajas:

- Por cada instructor se repite *building* y *budget* (redundancia)
- No se puede crear un departamento sin profesor

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

inst_dept (ID, name, salary, dept name, building, budget)



Características de un buen diseño relacional

Alternativas de diseño

Esquemas grandes

- Ventajas:
 - Tenerlos unidos ahorra operaciones de JOIN
- Desventajas:
 - Por cada instructor se repite *building* y *budget* (redundancia)
 - No se puede crear un departamento sin profesor

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

inst_dept (ID, name, salary, dept name, building, budget)

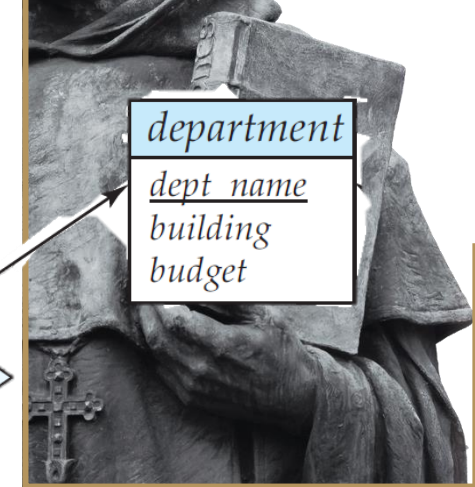
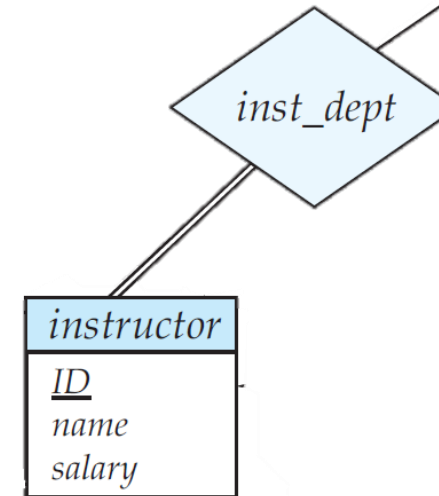


Características de un buen diseño relacional

Alternativas de diseño

Esquemas pequeños

- Evitan la repetición de información
- Difíciles de reconocer en una base de datos del mundo real
 - ¿La repetición es patrón o coincidencia?
- Fáciles de reconocer en reglas de negocio
 - P.ej.: “La universidad requiere que cada departamento tenga únicamente un edificio y un único valor de presupuesto”





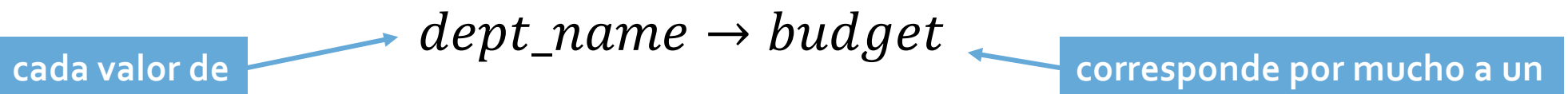
Características de un buen diseño relacional

Dependencia funcional

“La universidad requiere que cada departamento tenga únicamente un edificio y un único valor de presupuesto”

A cada departamento le corresponde por mucho un presupuesto, inclusive donde *dept_name* no es llave primaria

Si hay un esquema (*dept_name*, *budget*) entonces *dept_name* puede ser PK





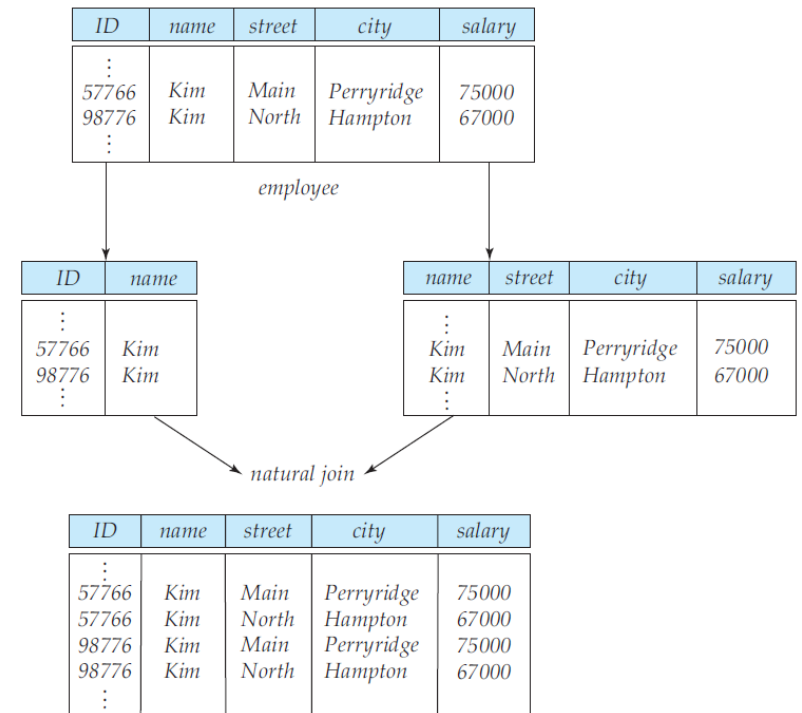
Características de un buen diseño relacional

Dependencia funcional

Los esquemas se pueden partir o descomponer

La descomposición de un esquema puede ser

- con pérdidas
- sin pérdidas





Características de un buen diseño relacional

Análisis de anomalías

ACTIVIDAD: Identificar dependencias funcionales en las siguientes relaciones:

- **item_factura**(num_factura, num_item, id_cliente, nom_cliente, num_producto, nom_producto, unidades_vendidas, valor_producto, valor_total_factura, fecha_factura)
- **estudiante**(cod_estudiante, nombre, fechaNacimiento, semestre, prom_por_semestre)
- **carrito_mercado**(id_carrito, id_cliente, id_producto, cantidad, fecha_vigencia_carrito)

Unidad de Educación Continua y Consultoría
construimos país desde

#URSolucionesInnovadoras
#URConsultoría



@RosarioContinua



/EduContinuaURosario



@RosarioContinua