Contraint types

→ PRIMARY KEY – The unique identifier for the current row, which is always NOT NULL

→ REFERENCES (FOREIGN KEY) – A key that refers to a primary key in a different table, signifying their relationship to each other

→ UNIQUE – Two cells in this Column cannot be the same

→ NOT NULL – This attribute HAS to be specified

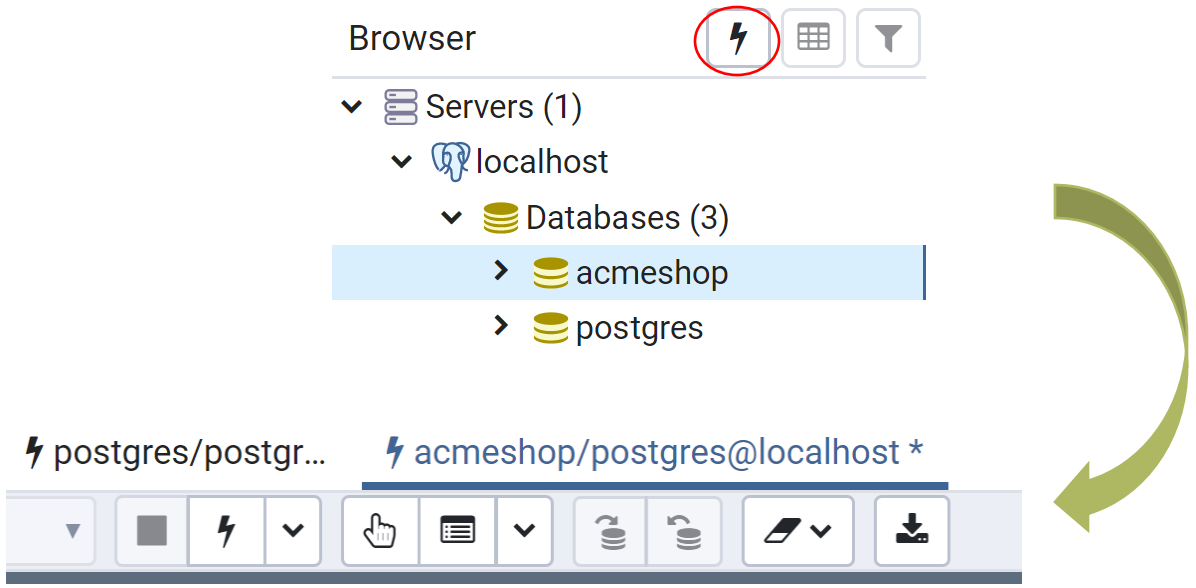
Constraints and Data types in SQL

```
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
);
```

Creating a Database

- → PostgreSQL only allows connections to one database at a time, and does not allow switching between them.
- → This is not normal, and usually you can “use database” to attach to another database.
- → Because of this restriction, run the command, pick the newly created database, and start a new SQL window.

```
-- creating a database
create database AcmeShop;
```



Deleting a Database

```
drop database AcmeShop;
```

Active connections to a database will block a deletion attempt!

Tables and relationships

Customers Table

	id [PK] integer	username character varying (50)	password character varying (50)	email character varying (355)	created_on timestamp without time zone	last_login timestamp without time zone
1	1	John	myPassW0rd	john@acme.com	2020-02-13 10:40:41.660572	[null]
2	2	Anne	SomePassword	anne@acme.com	2020-02-13 10:40:41.660572	[null]

Products Table

	id [PK] integer	name character varying (150)	price real
1	1	Samsung Galaxy S20	7799.95
2	2	Samsung Galaxy S20 - Leathe…	799.95
3	3	iPhone 11 Pro	8899
4	4	iPhone 11 Pro - Leather Cover	399.5
5	5	Huawai P30 Lite	1664.5
6	6	Huawai P30 - Leather Cover	1664.5

Orders Table

	id [PK] integer	order_number character (10)	customer_id integer
1	1	DA-0001234	1
2	2	DA-0001235	1
3	3	DE-0001236	2
4	4	DE-0001237	2

Order_Lines Table

	id [PK] integer	order_id integer	product_id integer	amount integer
1	1	1	1	2
2	2	1	2	2
3	3	1	5	1
4	4	3	3	2
5	5	3	4	1
6	6	4	1	1

Creating Tables with Relationships

```
CREATE TABLE account(
   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
);

CREATE TABLE blog_entries(
    id serial PRIMARY KEY,
    header varchar(255) NOT NULL,
    body TEXT NOT NULL,
    created_by integer NOT NULL REFERENCES account(id)
);
```

Constraint Enforcement

```sql
INSERT INTO account (username, password, email, created_on)
VALUES ('John', 'myPassW0rd', 'john@acme.com', NOW()); -- becomes user 1
```

```sql
INSERT INTO blog_entries (header, body, created_by)
VALUES ('My article', 'my body text', 1); -- works!
```

```sql
INSERT INTO blog_entries (header, body, created_by)
VALUES ('My article', 'my body text', 6); -- ERROR!
```

```
ERROR:  insert or update on table "blog_entries" violates foreign key constraint "blog_entries_created_by_fkey"
DETAIL:  Key (created_by)=(6) is not present in table "account".
SQL state: 23503
```

Deleting tables, and tables with constraints

```
CREATE TABLE account(
   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
);

CREATE TABLE blog_entries(
    id serial PRIMARY KEY,
    header varchar(255) NOT NULL,
    body TEXT NOT NULL,
    created_by integer NOT NULL REFERENCES account(id)
);

drop table account;
```

→ Delete in order, where non of a tables attributes has a constraint from another table

```
ERROR:  cannot drop table account because other objects depend on it
DETAIL:  constraint blog_entries_created_by_fkey on table blog_entries depends on table account
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
SQL state: 2BP01
```

Altering tables and deleting them

```
-- creating and altering tables
CREATE TABLE my_table(
   id serial PRIMARY KEY,
   my_attribute VARCHAR (50) UNIQUE NOT NULL
);

ALTER TABLE my_table ADD COLUMN my_new_coulmn TIMESTAMP;

ALTER TABLE my_table ALTER COLUMN my_new_coulmn TYPE varchar(50);

ALTER TABLE my_table DROP COLUMN my_new_coulmn;

DROP TABLE my_table;
```

Inserts (CRUD)

```
INSERT INTO account (username, password, email, created_on)
VALUES ('John', 'myPassW0rd', 'john@acme.com', NOW());

INSERT INTO account (username, password, email, created_on)
VALUES ('Anne', 'myPassW0rd', 'anne@acme.com', NOW());
```

	user_id [PK] integer	username character varying (50)	password character varying (50)	email character varying (355)	created_on timestamp without time zone	last_login timestamp without time zone
1	4	John	myPassW0rd	john@acme.com	2020-02-06 11:43:44.158522	[null]
2	5	Anne	myPassW0rd	anne@acme.com	2020-02-06 11:43:44.158522	[null]

Selects (CRUD)

```sql
SELECT * FROM account;

SELECT username, created_on FROM account WHERE email = 'john@acme.com';

SELECT username, created_on FROM account WHERE email LIKE '%anne%';
```

	user_id [PK] integer	username character varying (50)	password character varying (50)	email character varying (355)	created_on timestamp without time zone	last_login timestamp without time zone
1	1	John	myPassW0rd	john@acme.com	2020-02-06 11:41:11.729203	[null]

Updates (CRUD)

```
UPDATE account SET password = 'newPassW0rd' WHERE username = 'Anne';
```

	user_id [PK] integer	username character varying (50)	password character varying (50)	email character varying (355)	created_on timestamp without time zone	last_login timestamp without time zone
1	1	John	myPassW0rd	john@acme.com	2020-02-06 11:41:11.729203	[null]
2	2	Anne	newPassW0rd	anne@acme.com	2020-02-06 11:42:13.27506	[null]

Delete (CRU**D**)

```
DELETE FROM account WHERE email = 'john@acme.com';
```

	user_id [PK] integer	username character varying (50)	password character varying (50)	email character varying (355)	created_on timestamp without time zone	last_login timestamp without time zone
1	5	Anne	myPassW0rd	anne@acme.com	2020-02-06 11:43:44.158522	[null]

BEWARE of doing this: `DELETE FROM account;`

Querying Your Result Set - Nested Queries

```
CREATE TABLE account(
   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
);

CREATE TABLE blog_entries(
    id serial PRIMARY KEY,
    header varchar(255) NOT NULL,
    body TEXT NOT NULL,
    created_by integer NOT NULL REFERENCES account(id)
);
```

```
select * from account, blog_entries
where blog_entries.created_by = account.id;
```

	id integer	username character varying (50)	password character varying (50)	email character varying (355)	created_on timestamp without time zone	last_login timestamp witho
1	1	John	myPassW0rd	john@acme.com	2020-02-13 11:55:37.125376	[null]
2	1	John	myPassW0rd	john@acme.com	2020-02-13 11:55:37.125376	[null]
3	1	John	myPassW0rd	john@acme.com	2020-02-13 11:55:37.125376	[null]
4	2	Anne	SomePassword	anne@acme.com	2020-02-13 11:55:42.35475	[null]
5	1	John	myPassW0rd	john@acme.com	2020-02-13 11:55:37.125376	[null]

```
select username, email, created_by from (
    select * from account, blog_entries
    where blog_entries.created_by = account.id
) as result_set where created_by = 2;
```

	username character varying (50)	email character varying (355)	created_by integer
1	Anne	anne@acme.com	2

Join types

- → INNER JOIN
  - → For each row R1 of T1, the joined table has a row for each row in T2 that satisfies the join condition with R1.
- → LEFT OUTER JOIN
  - → First, an inner join is performed. Then, for each row in T1 that does not satisfy the join condition with any row in T2, a joined row is added with null values in columns of T2. Thus, the joined table always has at least one row for each row in T1.
- → RIGHT OUTER JOIN
  - → First, an inner join is performed. Then, for each row in T2 that does not satisfy the join condition with any row in T1, a joined row is added with null values in columns of T1. This is the converse of a left join: the result table will always have a row for each row in T2.
- → FULL OUTER JOIN
  - → First, an inner join is performed. Then, for each row in T1 that does not satisfy the join condition with any row in T2, a joined row is added with null values in columns of T2. Also, for each row of T2 that does not satisfy the join condition with any row in T1, a joined row with null values in the columns of T1 is added.
- → Source: https://www.postgresql.org/docs/9.2/queries-table-expressions.html

Inner Join

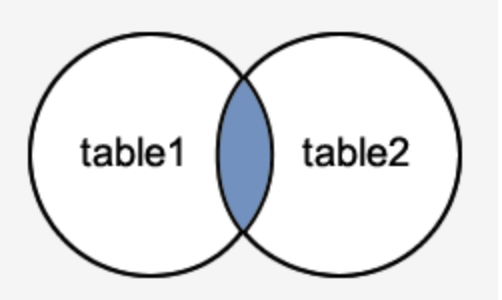
```
SELECT * FROM t1 INNER JOIN t2 ON t1.num = t2.num;
```

```
SELECT * FROM t1,t2 WHERE t1.num = t2.num;
```

num integer	name character varying (10)	num integer	value character varying (10)
1	a	1	xxx
3	c	3	yyy

```
SELECT * FROM t1 INNER JOIN t2 USING (num);
```

num integer	name character varying (10)	value character varying (10)
1	a	xxx
3	c	yyy



T1 table

num integer	name character varying (10)
1	a
2	b
3	c

T2 table

num integer	value character varying (10)
1	xxx
3	yyy
5	zzz

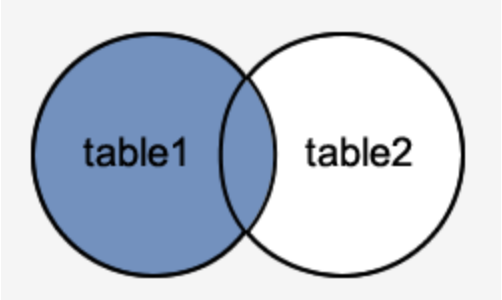
Left Outer Join

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num;
```

num integer	name character varying (10)	num integer	value character varying (10)
1	a	1	xxx
2	b	[null]	[null]
3	c	3	yyy

```
SELECT * FROM t1 LEFT JOIN t2 USING (num);
```

num integer	name character varying (10)	value character varying (10)
1	a	xxx
2	b	[null]
3	c	yyy



T1 table

num integer	name character varying (10)
1	a
2	b
3	c

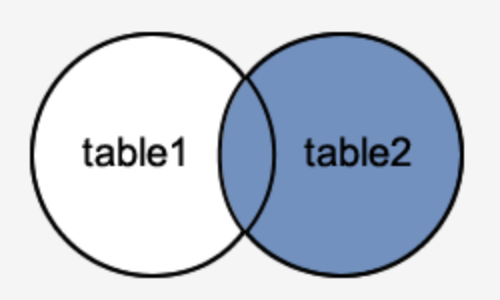
T2 table

num integer	value character varying (10)
1	xxx
3	yyy
5	zzz

Right Outer Join

```
SELECT * FROM t1 RIGHT JOIN t2 ON t1.num = t2.num;
```

num integer	name character varying (10)	num integer	value character varying (10)
1	a	1	xxx
3	c	3	yyy
[null]	[null]	5	zzz



T1 table

num integer	name character varying (10)
1	a
2	b
3	c

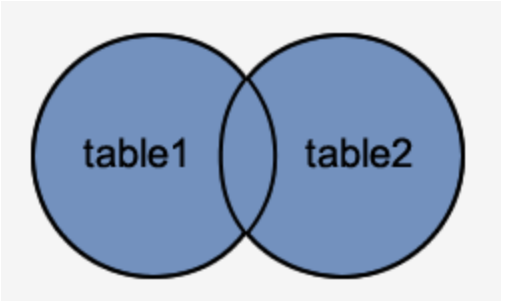
T2 table

num integer	value character varying (10)
1	xxx
3	yyy
5	zzz

Full Outer Join

```
SELECT * FROM t1 FULL JOIN t2 ON t1.num = t2.num;
```

num integer	name character varying (10)	num integer	value character varying (10)
1	a	1	xxx
2	b	[null]	[null]
3	c	3	yyy
[null]	[null]	5	zzz



T1 table

num integer	name character varying (10)
1	a
2	b
3	c

T2 table

num integer	value character varying (10)
1	xxx
3	yyy
5	zzz

Views – Creating Virtual Tables

```
CREATE VIEW MyCustomView AS
    SELECT email, username FROM account, blog_entries
    WHERE blog_entries.created_by = account.id;

SELECT * FROM MyCustomView;
```

Query defining the view content

	email character varying (355)	username character varying (50)
1	john@acme.com	John
2	john@acme.com	John
3	john@acme.com	John
4	anne@acme.com	Anne
5	john@acme.com	John

- Can join and simplify multiple tables into a view.
- Result of the query in the view is dynamically updated, when new database content id added.
- Can hide complexity of data
- Takes very little space as only the query is stored