

Creating tables and inserted data using Mockaroo:

Table person:

```
1  create table person_data1 (
2      user_id BIGSERIAL NOT NULL PRIMARY KEY,
3      first_name VARCHAR(50) NOT NULL,
4      last_name VARCHAR(50) NOT NULL,
5      email VARCHAR(150) NOT NULL,
6      password VARCHAR(50) NOT NULL
7  );
```

```
abilmansurtoiganbay=# \d person_data1
Table "public.person_data1"
 Column |      Type       | Collation | Nullable | Default
-----+---------------+-----+-----+
 user_id | bigint        |          | not null | nextval('person_data1_user_id_seq'::regclass)
 first_name | character varying(50) |          | not null |
 last_name | character varying(50) |          | not null |
 email | character varying(150) |          | not null |
 password | character varying(50) |          | not null |
Indexes:
 "person_data1_pkey" PRIMARY KEY, btree (user_id)
Referenced by:
 TABLE "priority" CONSTRAINT "connection_with_person_data" FOREIGN KEY (person_id) REFERENCES person_data1(user_id)
 TABLE "category" CONSTRAINT "connection_with_person_data" FOREIGN KEY (person_id) REFERENCES person_data1(user_id)
 TABLE "task" CONSTRAINT "fk_task" FOREIGN KEY (user_id) REFERENCES person_data1(user_id)
```

Table task:

```
create table task (
    task_id BIGSERIAL NOT NULL PRIMARY KEY,
    task_name VARCHAR(250) NOT NULL,
    task_state VARCHAR(50) NOT NULL,
    task_date DATE,
    priority_id BIGINT,
    category_id BIGINT,
    user_id BIGINT NOT NULL,
    CONSTRAINT FK_task FOREIGN KEY(user_id) REFERENCES person_data1(user_id)
);
```

```
abilmansurtoiganbay=# \d task;
Table "public.task"
 Column |      Type       | Collation | Nullable | Default
-----+---------------+-----+-----+
 task_id | bigint        |          | not null | nextval('task_task_id_seq'::regclass)
 task_name | character varying(250) |          | not null |
 task_state | character varying(50) |          | not null |
 task_date | date          |          | not null |
 priority_id | bigint        |          |          |
 category_id | bigint        |          |          |
 user_id | bigint        |          | not null |
Indexes:
 "task_pkey" PRIMARY KEY, btree (task_id)
Foreign-key constraints:
 "fk_task" FOREIGN KEY (user_id) REFERENCES person_data1(user_id)
Referenced by:
 TABLE "priority" CONSTRAINT "connection_with_task" FOREIGN KEY (task_id) REFERENCES task(task_id)
 TABLE "category" CONSTRAINT "connection_with_task_data" FOREIGN KEY (task_id) REFERENCES task(task_id)
```

Priority table:

```
1  create table priority (
2      priority_id BIGSERIAL NOT NULL PRIMARY KEY,
3      priority_name TEXT NOT NULL,
4      priority_color VARCHAR(50),
5      person_id BIGSERIAL NOT NULL,
6      task_id BIGSERIAL NOT NULL,
7      CONSTRAINT connection_with_task FOREIGN KEY (task_id) REFERENCES task(task_id),
8      CONSTRAINT connection_with_person_data FOREIGN KEY (person_id) REFERENCES person_data1(user_id)
9  );
```

```
[abilmansurtoiganbay-# \d priority
Table "public.priority"
 Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----+
priority_id | bigint | | not null | nextval('priority_priority_id_seq'::regclass)
priority_name | text | | not null |
priority_color | character varying(50) | | |
person_id | bigint | | not null | nextval('priority_person_id_seq'::regclass)
task_id | bigint | | not null | nextval('priority_task_id_seq'::regclass)
Indexes:
"priority_pkey" PRIMARY KEY, btree (priority_id)
Foreign-key constraints:
"connection_with_person_data" FOREIGN KEY (person_id) REFERENCES person_data1(user_id)
"connection_with_task" FOREIGN KEY (task_id) REFERENCES task(task_id)
-
```

Category table:

```
create table category (
    category_id BIGSERIAL NOT NULL PRIMARY KEY,
    category_name TEXT NOT NULL,
    person_id BIGINT NOT NULL,
    task_id BIGINT NOT NULL,
    CONSTRAINT connection_with_person_data FOREIGN KEY (person_id) REFERENCES person_data1(user_id),
    CONSTRAINT connection_with_task_data FOREIGN KEY (task_id) REFERENCES task(task_id)
);
```

```
[abilmansurtoiganbay-# \d category
Table "public.category"
 Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----+
category_id | bigint | | not null | nextval('category_category_id_seq'::regclass)
category_name | text | | not null |
person_id | bigint | | not null |
task_id | bigint | | not null |
Indexes:
"category_pkey" PRIMARY KEY, btree (category_id)
Foreign-key constraints:
"connection_with_person_data" FOREIGN KEY (person_id) REFERENCES person_data1(user_id)
"connection_with_task_data" FOREIGN KEY (task_id) REFERENCES task(task_id)
```

Created tables and relations between them:

List of relations			
Schema	Name	Type	Owner
public	category	table	postgres
public	category_category_id_seq	sequence	postgres
public	person_data1	table	postgres
public	person_data1_user_id_seq	sequence	postgres
public	priority	table	postgres
public	priority_person_id_seq	sequence	postgres
public	priority_priority_id_seq	sequence	postgres
public	priority_task_id_seq	sequence	postgres
public	task	table	postgres
public	task_task_id_seq	sequence	postgres

Answers:

1. Primary Key - это уникальное значение в таблице, не может быть null, может быть только один PK в одной таблице
Foreign Key - это уникальное значение которое мы используем для связи данных с другой таблицей ссылаясь на Primary Key нужной таблицы.
2. Объединение данных это объединение данных в нескольких разных таблицах, через связанные колонны. Происходит это используя команды INNER JOIN, RIGHT JOIN, LEFT JOIN, FULL JOIN.

Examples:

INNER JOIN:

```
1 SELECT first_name, last_name, email, task_name, task_state, task_date
2 FROM person_data1
3 INNER JOIN task
4 ON person_data1.user_id = task.task_id;
```

#	first_name	last_name	email	task_name
1	Celle	Lemerle	clemeler0@omniture.com	Meth
2	Agretha	Greger	agreger1@fastcompany.com	Marvin Gaye: What's Going On
3	Lauretta	Owain	lowain2@zdn.net	Mercy Streets
4	Stephannie	Brackstone	sbrackstone3@webnode.com	Cold Mountain
5	Jilly	Creer	jcreer4@bigcartel.com	I Am the Law
6	Magda	Vittet	mvittet5@google.com.au	Whirlpool
7	Regen	Poge	rpose6@telegraph.co.uk	Territories
8	Luciana	Fiddiman	lfiddiman7@exblog.jp	Blackout
9	Andrea	Maiden	amaiden8@engadget.com	Harry and the Hendersons
10	Celka	Radley	cradley9@harvard.edu	As Tears Go By (Wong gok ka moon)
11	Vina	Smedmoor	vsmedmoora@nytimes.com	Revenge of the Zombies
12	Kali	Franzolini	kfranzolinib@desdev.cn	Back in Business

RIGHT JOIN

```
1 SELECT first_name, last_name, priority_name, priority_color
2 FROM person_data1
3 RIGHT JOIN priority
4 ON person_data1.user_id = priority.person_id;
```

	first_name character varying (50)	last_name character varying (50)	priority_name text	priority_color character varying (50)
1	Noreen	Headingham	aliquam non mauris morbi	false
2	Dex	Kuhl	premium iaculis justo	true
3	Petronella	Recher	diam vitae quam suspendisse potenti	true
4	Rozella	Bocking	tellus	true
5	Holly	Lingard	nullam varius nulla facilisi cras	false
6	Renee	Colley	ipsum primis in faucibus orci	true
7	Sutherlan	Danilovich	adipiscing elit proin	false

LEFT JOIN

```

1  SELECT first_name, last_name, category_name
2  FROM category
3  RIGHT JOIN person_data1
4  ON category.person_id = person_data1.user_id;

```

	first_name character varying (50)	last_name character varying (50)	category_name text
1	Chrissie	Axe	ac neque
2	Vincenty	Imason	maecenas tincidunt lacus at
3	Thaxter	Tale	id
4	Dotty	McGeorge	ultrices libero
5	Ashley	Zincke	vel nulla eget
6	Patton	Ferraro	adipiscing molestie hendrerit at

FULL JOIN

```

1  SELECT first_name, last_name, task_name
2  FROM person_data1
3  FULL JOIN task
4  ON person_data1.user_id = task.user_id;

```

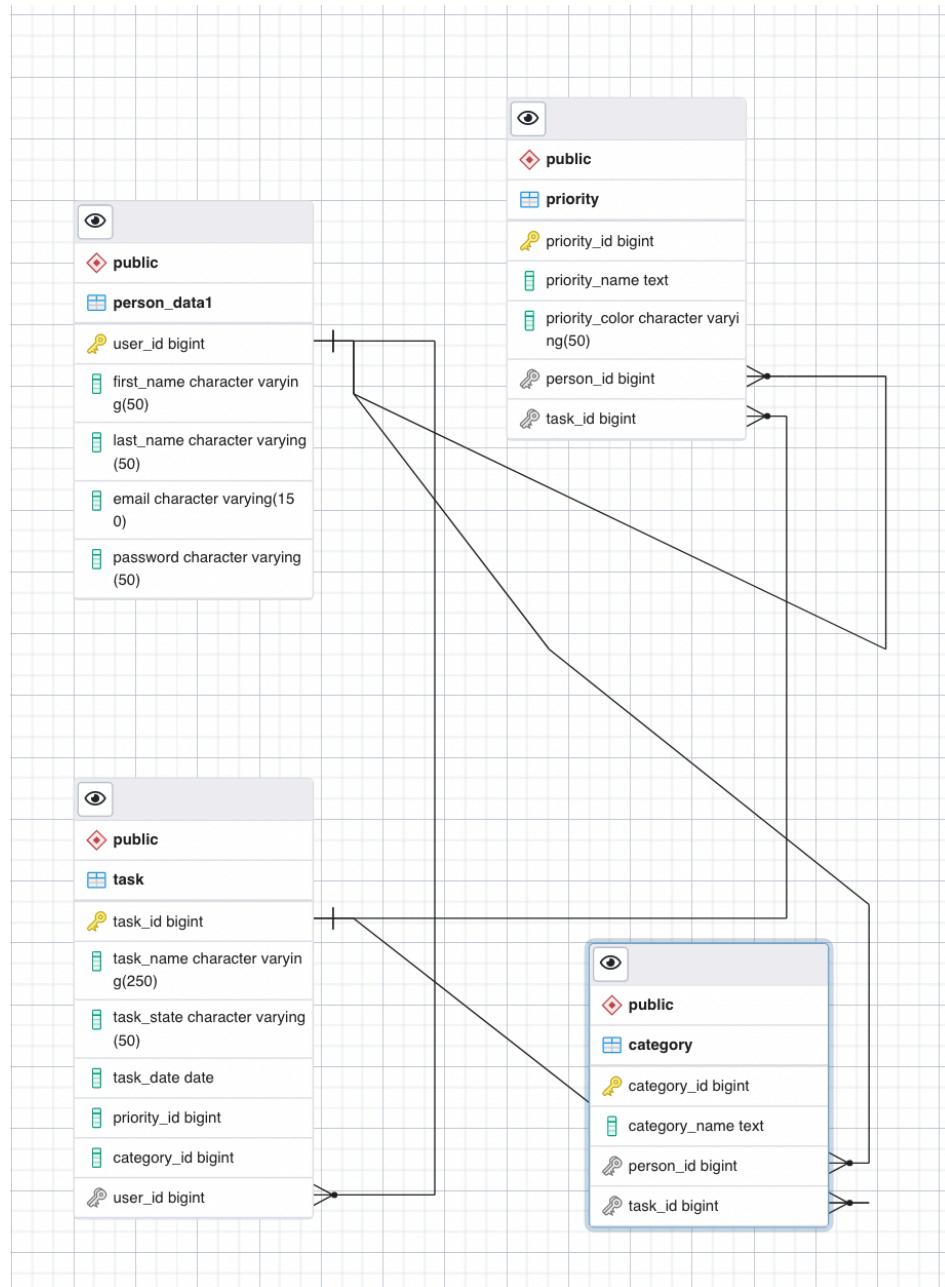
	first_name character varying (50)	last_name character varying (50)	task_name character varying (250)
1	Read-only column E.....	Maddison	Meth
2	Devondra	Elcomb	Marvin Gaye: What's Going On
3	Barrett	Billin	Mercy Streets
4	Hayward	Philson	Cold Mountain
5	Dunstan	Hayselden	I Am the Law
6	Dillon	Raistrick	Whirlpool
7	Magda	Vittet	Territories

3. Есть 3 вида связи таблиц это One to One, One to Many, Many to Many.

One to one: это когда одна запись в первой таблице может иметь связь только с одной записью в второй таблице. Например один работник может иметь только одну корпоративную машину, или муж может иметь только одну жену.

One to many: это когда одна запись в первой таблице может иметь связь с несколькими записями в таблице. Например в нашем случае person может иметь несколько записей в таблице task.

Many to many: Может быть когда несколько записей в первой таблице может иметь связь с несколькими записями во второй таблице. Например у работника может быть несколько должностей, а на одной должности могут быть несколько работников.



4. Агреггационные функции: SUM, COUNT, MIN, MAX, AVG.

```
1  SELECT COUNT(*) FROM priority WHERE priority_color = 'true';
```

	count	bigint
1	499	

5. GROUP BY мы используем эту команду для объединения нужных строк по нужному нам условию. Например мы можем посчитать сколько танков есть у каждого человека и вывести по уменьшающему порядку.

```

1 SELECT person_data1.first_name, COUNT(task.task_id) AS numberoftasks FROM person_data1
2 RIGHT JOIN task ON person_data1.user_id = task.user_id
3 GROUP BY first_name
4 ORDER BY numberoftasks DESC;
```

	first_name	numberoftasks
1	Nyssa	6
2	Etti	6
3	Elga	5
4	Aubrey	5
5	Laurel	5
6	Bennie	5
7	Gwendolen	5
8	Leoline	4
9	Moyna	4
10	Aveline	4

6. Первая нормальная форма: каждая строка должна иметь одно значение. Не так чтобы был массив с данными в одной ячейке. Колонны в таблицах не должны повторяться.

Вторая нормальная форма: столбцы(не первичный ключ) должны зависеть от первичного ключа. Может содержать столбцы которые могут изменится при изменении другого столбца.

Третья нормальная форма: столбцы(не первичный ключ) должны зависеть ТОЛЬКО от первичного ключа. Не может содержать столбцы которые могут изменится при изменении другого столбца.

Для перехода разделяем таблицы.

7. Materialized view - создаем view (физическое) к которому мы сможем потом обращаться. Через CREATE MATERIALIZED VIEW... похож на таблицу. После изменений в таблицах которые использованы в view можно его просто обновить через команду REFRESH. View - создаем view который является абстрактным которое выведет что просим но чтобы дальше с ним работать надо его пересоздавать через CREATE VIEW...

8. В sql можно объединить от 2х и более таблиц. Чередуя команды JOIN, если 3 таблицы то 2 JOIN, если 4 таблицы то 3 JOIN и так далее. Меньше двух таблиц нельзя так как это не имеет смысла, в этом случае надо работать с одной таблицей.

9. Сделать запрос где есть:

- выборка: функция агрегации + обычные поля

См. Вопрос номер 4.

- группировка по нескольким полям

См. Вопрос номер 5.

- join нескольких таблиц

```

1 SELECT person_data1.first_name, person_data1.last_name, task.task_name, priority.priority_name, priority.priority_color
2 FROM person_data1 RIGHT JOIN task ON person_data1.user_id = task.user_id
3 RIGHT JOIN priority ON task.task_id = priority.task_id;

```

	first_name character varying (50)	last_name character varying (50)	task_name character varying (250)	priority_name text
1	Benedetta	Titterton	Don't Change Your Husband	aliquam non mauris morbi
2	Barrett	Billin	Chastity Bites	pretium iaculis justo
3	Jozef	Trim	Underworld: Evolution	diam vitae quam suspendisse pc
4	Fons	Ridett	Newsfront	tellus
5	Shaun	Dowdall	Robin and the 7 Hoods	nullam varius nulla facilisi cras
6	Andria	Donke	Christmas Carol, A	ipsum primis in faucibus orci

- условие where

```

1 SELECT person_data1.first_name, person_data1.last_name, task.task_name, priority.priority_name, priority.priority_color
2 FROM person_data1 RIGHT JOIN task ON person_data1.user_id = task.user_id
3 RIGHT JOIN priority ON task.task_id = priority.task_id
4 WHERE priority.priority_color = 'true'; |

```

	first_name character varying (50)	last_name character varying (50)	task_name character varying (250)	priority_name text	priority_color character varying
1	Barrett	Billin	Chastity Bites	pretium iaculis justo	true
2	Jozef	Trim	Underworld: Evolution	diam vitae quam suspendisse potenti	true
3	Fons	Ridett	Newsfront	tellus	true
4	Andria	Donke	Christmas Carol, A	ipsum primis in faucibus orci	true
5	Fae	Swash	Fathers' Day	ut odio cras mi pede	true
6	Mandie	O'Crowley	In & Out	sodales scelerisque	true
7	andi	Teresia	Eight Miles High (Das wilde Leben)	rutrum neque aenean auctor	true
8	Quintina	Djurdevic	Punksters & Youngsters (Punk - Tauti joka ei tapa)	augue	true
9	Chrystal	Engledow	Black Dahlia, The	phasellus	true

- сортировка результата

```

1 SELECT person_data1.first_name, person_data1.last_name, task.task_name, priority.priority_name, priority.priority_color
2 FROM person_data1 RIGHT JOIN task ON person_data1.user_id = task.user_id
3 RIGHT JOIN priority ON task.task_id = priority.task_id
4 WHERE priority.priority_color = 'true'
5 ORDER BY person_data1.last_name ASC;

```

	first_name character varying (50)	last_name character varying (50)	task_name character varying (250)	priority_name text	priority_color character varying
1	Elfie	Abazi	Pillars of the Earth, The	lorem vitae mattis nibh ligula	true
2	Gasparo	Abbots	Jakob the Liar	vel pede	true
3	Gasparo	Abbots	Jakob the Liar	mollis molestie lorem quisque ut	true
4	Hamel	Androletti	ZMD: Zombies of Mass Destruction	amet	true
5	Hamel	Androletti	Rat Race	nibh in quis	true
6	Hamel	Androletti	ZMD: Zombies of Mass Destruction	velit	true
7	Lainey	Annon	Jew Süss (Jud Süß)	pellentesque quisque	true
8	Britta	Anwyl	And Your Mother Too (Y tu mamá también)	eu sapien	true

10. SELECT, INSERT, DELETE, UPDATE

11. ALTER INDEX index_name ON table_name.column_name SET (something);

12. Мы используем индексы чтобы получать данные быстрее при поиске, обычно программы перебирает через SELECT все, но при поиске в индексированных данных команда происходит быстрее так как когда мы создаем индекс они хранятся в бинарном дереве. Можно использовать команду EXPLAIN чтобы посмотреть скорость и как проводилась команда.

13. COALESCE вернет первое не NULL значение в ней.

```
1 SELECT COALESCE((SELECT COUNT(task_date) FROM task WHERE task_name = 'Abilmansur'), 0);
```

Записей 'Abilmansur' нет в таблице тасков и внутренний селект должен выдать нулл, но после COALESCE он выдает первое не нулл значение а это 0.

14. Да существует, для этого мы используем метод CAST(initial AS needed)

```
1 | SELECT * FROM task WHERE task_date = CAST('06-06-2022' AS DATE);
```

	task_id [PK] bigint	task_name character varying (250)	task_state character varying (50)	task_date date	priority_id bigint	category_id bigint	user_id bigint
1	6	Whirlpool	true	2022-06-06	471	341	526
2	17	Outside the Law	true	2022-06-06	945	416	563

15. Function:

```
CREATE FUNCTION get_tasks_by_date(date_from varchar(10), date_to varchar(10)) RETURNS int
LANGUAGE plpgsql
as
$$
DECLARE
date_count integer;
BEGIN
SELECT count(*)
into date_count
from task
where task_date between CAST(Price_from AS DATE)
and CAST(Price_to AS DATE);
RETURN date_count;
END; $$;
```

CREATE FUNCTION

Query returned successfully in 38 msec.

```
1  SELECT get_tasks_by_date('01-01-2020','01-01-2024');
```

	get_tasks_by_date	lock
	integer	
1		
	510	

Trigger:

```
CREATE TABLE changed_tasks (
    id INT GENERATED ALWAYS AS IDENTITY,
    task_id INT NOT NULL,
    task_name VARCHAR(200) NOT NULL,
    changed_on TIMESTAMP(6) NOT NULL
);
```

```
1 CREATE OR REPLACE FUNCTION log_task_changes()
2     RETURNS TRIGGER
3     LANGUAGE PLPGSQL
4     AS
5     $$
6 ▼ BEGIN
7 ▼     IF NEW.task_name <> OLD.task_name THEN
8         INSERT INTO changed_tasks(task_id,task_name,changed_on)
9             VALUES(OLD.task_id,OLD.task_name,now());
10    END IF;
11
12    RETURN NEW;
13 END;
14 $$

1
2 CREATE TRIGGER task_name_changes
3     BEFORE UPDATE
4     ON task
5     FOR EACH ROW
6     EXECUTE PROCEDURE log_task_changes();

1 UPDATE task
2 SET task_name = 'newTaskName'
3 WHERE task_id = 1;

^
SELECT * FROM changed_tasks;
```

	id integer	task_id integer	task_name character varying (200)	changed_on timestamp without time zone
1	1	1	Meth	2022-05-03 20:18:13.703723

16. Transactions:

```
1 BEGIN;
2 INSERT INTO person_data1 (user_id, first_name, last_name, email, password)
3 VALUES (1001, 'Abilmansur', 'Toiganbay', 'abilamansur.t@gmail.com', 'postgresql');
4 COMMIT;
```

Транзакция позволяет нам работать и изменять базу используя команды BEGIN, ROLLBACK, COMMIT.

BEGIN - объявление начала транзакции все изменения в базе будут запомнены ждать завершения.

ROLLBACK - Если мы еще не закомитили изменения то можно откатить изменения.

COMMIT - завершаем транзакцию внесены в базу данных.

17. ACID - ATOMIC, CONSISTENT, ISOLATED, DURABLE. По этим принципам работают транзакции в PGSQl.

ATOMIC - Либо выполняется все, либо ничего.

CONSISTENT - Записываемые данные должны соответствовать данным нами правилам, если не подходит то мы отменяем то есть откатываем транзакцию.

ISOLATED - отвечает за целостность данных используемых во время транзакции, например заблокировать одну строку или целую таблицу при выполнении транзакции, так как возможны баги при параллельной работе других пользователей в таблице.

DURABLE - долговечность отвечает за то что введенные данные будут постоянно хранится в БД.