id ▲ 1	name	dob	group_id
1	Jim	1975-05-07	1
2	Sergio	1997-03-04	1
3	Jackson	1996-04-06	2
10	Jeremy	1998-07-08	2
11	Jack	1994-05-06	3
15	Mickey	2001-02-03	-1
20	Alex	1993-08-09	3
30	Marion	1997-07-07	3
31	Mario	1996-03-04	4
41	Alexander I	1997-04-05	5
42	Alexander II	1985-01-01	5
<null>	<null>	<null>	<null>

```
create table if not exists person
(
    id        integer,
    name      text not null,
    dob       date not null,
    group_id integer
);
```

<null> Alexander II <null> <null>

???

```
be5.public> INSERT INTO public.person (id, name, dob, group_id) VALUES (null, 'Alexander II', null, null)
[2022-10-02 09:11:06] [23502] ERROR: null value in column "dob" violates not-null constraint
[2022-10-02 09:11:06] Detail: Failing row contains (null, Alexander II, null, null).
```

preventing storing data in a wrong way on early stage is a big win.

	id ▲ 1	name	dob
1	1	Jim	1975-05-07
2	2	Sergio	1997-03-04
3	3	Jackson	1996-04-06
4	10	Jeremy	1998-07-08
5	11	Jack	1994-05-06
6	15	Mickey	2001-02-03
7	20	Alex	1993-08-09
8	30	Marion	1997-07-07
9	31	Mario	1996-03-04
10	41	Alexander I	1997-04-05
11	42	Alexander II	1985-01-01
12	<null>	Alexander II	1991-08-07

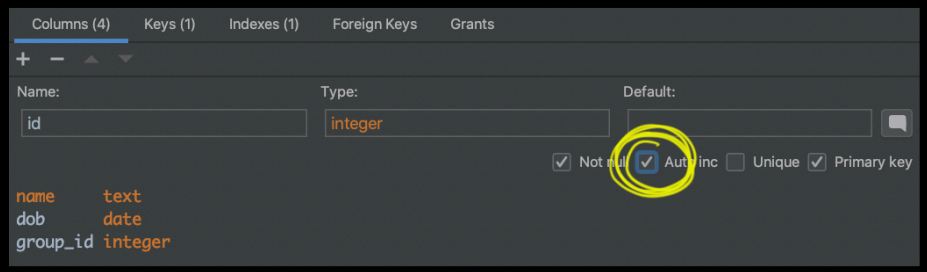
```
select * from person
where id=42;
```

id	name	dob	group_id
42	Alexander II	1985-01-01	5
42	Alexander II	1991-08-07	<null>

identifier should be unique!

when field is NOT NULL + UNIQUE => field can be used as PK

what DIFF NOT NULL + UNIQUE vs PRIMARY KEY




```
create sequence person_id_seq;

alter table person
    alter column id set default nextval('public.person_id_seq'::regclass);
```

sequence starts from 1

```
alter sequence person_id_seq restart 43;
```

	id ▲ 1	name	dob	group_id
1	1	Jim	1975-05-07	1
2	2	Sergio	1997-03-04	1
3	3	Jackson	1996-04-06	2
4	10	Jeremy	1998-07-08	2
5	11	Jack	1994-05-06	3
6	15	Mickey	2001-02-03	-1
7	20	Alex	1993-08-09	3
8	30	Marion	1997-07-07	3
9	31	Mario	1996-03-04	4
10	41	Alexander I	1997-04-05	5
11	42	Alexander II	1985-01-01	5
12	43	Alexander II	1991-08-07	<null>
13	<default>	A3	1992-03-04	<null>

[23505] ERROR: duplicate key value violates unique constraint "person_pkey"
Detail: Key (id)=(1) already exists.

	43	Alexander II	1991-08-07
	44	A3	1991-04-05

```
INSERT INTO public.person (id, name, dob, group_id, status) VALUES (DEFAULT, 'J5', '2022-10-02', null, DEFAULT)
```

id	name	dob	group_id	status
45	J5	2022-10-02	<null>	created

```
create table if not exists person
(
    id       serial
        primary key,
    name     text not null,
    dob      date not null,
    group_id integer,
    status   text default 'created'::text
);
```

```
id       integer default nextval('person_id_seq'::regclass)
```

```
INSERT INTO public.person (name, dob) VALUES ('J6', '2022-11-11');
```

45	J5	2022-10-02	<null>	created
46	J6	2022-11-11	<null>	created

by serial

by default

Foreign key

person → groupp

id	name	dob	group_id
1	Jim	1975-05-07	1
2	Sergio	1997-03-04	1
3	Jackson	1996-04-06	2
10	Jeremy	1998-07-08	2
11	Jack	1994-05-06	3
15	Mickey	2001-02-03	-1

id	name
1	BE1
2	BE2
3	BE3
4	BE4
5	BE5
6	FE1
7	FE2
8	FE3

is it valid? NO!!

only in our brain

```
select *
from person p
join groupp g on g.id = p.group_id
```

our table

Table: person

Comment:

Columns (5) | Keys (1) | Indexes (1) | Foreign Keys (1) | Grants

Name: person_groupp_id_fk (string ideal.)

Target table: groupp

Update rule: no action

Delete rule: no action

Deferrability: not deferrable

Columns: From: group_id To: id

field from "our table" are there any REQ? → IT MUST BE PK

```
create table if not exists groupp
(
    id      serial
        primary key,
    name    text,
    type_id integer
);
```

foreign keys 1

person_groupp_id_fk (group_id) → groupp (id)

just name

SQL Script

```
alter table person
    add constraint person_groupp_id_fk
        foreign key (group_id) references groupp (id);
```

6	15	Mickey	2001-02-03	11

[23503] ERROR: insert or update on table "person" violates foreign key constraint "person_groupp_id_fk"
Detail: Key (group_id)=(11) is not present in table "groupp".

transactions

from	to	amount	date
"Jim"	"Kate"	$100.00	<null>
Alex	Tom	<null>	13.10.22

bad data examples

customer ~~sales~~ → ~~customer~~ → ~~address~~ → ~~street~~

delete * from streets
insert into streets ( ) ...( )

use "cascade" extremely careful

deleted boolean

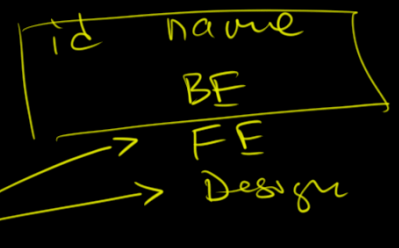
update users
set deleted = true
where id = 123

person

docsn't belong to person

group

id ▲ 1	name	dob	group_id
1	Jim	1975-05-07	1
2	Sergio	1997-03-04	1
3	Jackson	1996-04-06	2/null



Person — Person

Person — Group

1 — 1
1:1

1 — 0/1

Person < Group1 Group2

person < Group

we can't maintain that relation with that structure

1 — N
1:N

pg table_link

person

id ▲ 1	name	dob
1	Jim	1975-05-07
2	Sergio	1997-03-04
3	Jackson	1996-04-06

p_id	g_id
1	1
2	3
2	4

group

id ▲ 1	name
1	BE1
2	BE2
3	BE3
4	BE4

each row (p_id, g_id)

PK (p_id, g_id)

Columns (2) | Keys (1) | Indexes | For

```
p_id int -- part of primary key
g_id int -- part of primary key
```

Columns (2) | Keys (1) | Indexes | Foreign

```
pg_pk PRIMARY KEY pg (p_id, g_id)
```

```
pg_person_id_fk FOREIGN KEY (p_id) REFERENCES person (id)
pg_groupp_id_fk FOREIGN KEY (g_id) REFERENCES groupp (id)
```

```
create table pg
(
    p_id int
        constraint pg_person_id_fk
            references person,  (id)
    g_id int
        constraint pg_groupp_id_fk
            references groupp,  (id)
    constraint pg_pk
        unique (p_id, g_id)
);
```

person : group

≠

M : N

Many-to-Many

single responsibility

separation of concerns

person

id	name	dob
1	Jim	1975-05-07
2	Sergio	1997-03-04
3	Jackson	1996-04-06
10	Jeremy	1998-07-08

encapsulation

pg

p_id	g_id
1	1
2	3
2	4

relation

group

id	name
1	BE1
2	BE2
3	BE3
4	BE4

encapsulation

```
select p.id, p.name, g.id, g.name
from person p
join pg on p.id = pg.p_id
join groupp g on g.id = pg.g_id
```

p.id	p.name	g.id	g.name
1	Jim	1	BE1
2	Sergio	3	BE3
2	Sergio	4	BE4

always separate

- entities
- relations

person * group

person — group

person ⋹ group 1

group N

group * person

group — person

group ⋹ person 1

person N

p_id	g_id
1	1
2	3
2	4
43	4

```sql
select p.name
from person p
join pg on p.id = pg.p_id
join groupp g on g.id = pg.g_id
where g.name = 'BE4'
```

name
Sergio
Alexander II

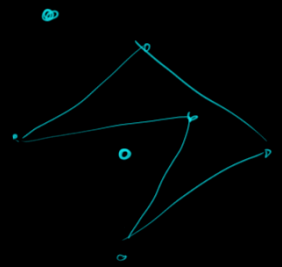
```sql
select g.name
from person p
join pg on p.id = pg.p_id
join groupp g on g.id = pg.g_id
where p.name = 'Sergio'
```

name
BE3
BE4

ANSI SQL 92

30 years old syntax

GPS



JSON

geometry
algebra
statistics

```sql
select p.name, count(p.name)
from person p
join pg on p.id = pg.p_id
join groupp g on g.id = pg.g_id
group by p.name
```

name	count
Sergio	2
Alexander II	1
Jim	1

```sql
select g.name, count(g.name)
from person p
join pg on p.id = pg.p_id
join groupp g on g.id = pg.g_id
group by g.name
```

name	count
BE1	1
BE3	1
BE4	2

50m records → w/o index 3-5 min

with index 100 ms

person_name	amount
Jim	100
John	100

Jim 120
John 80

we need to run them
in transaction
→ whether ALL
→ whether NOTHING

company has a budget of 200

```
update salary set amount = 120 where user = 'Jim'
update salary set amount = 80 where user = 'John'
```

conn. broken here
constraint violation exception

=> Jim 120
John 100

→ we will not able to understand what's wrong immediately

→ we will run out of budget.

conn.setAutoCommit (true)

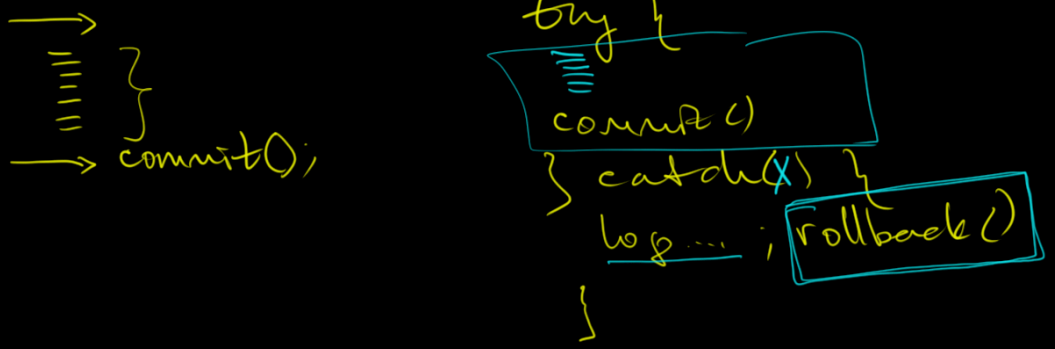
after each statement data persists into db

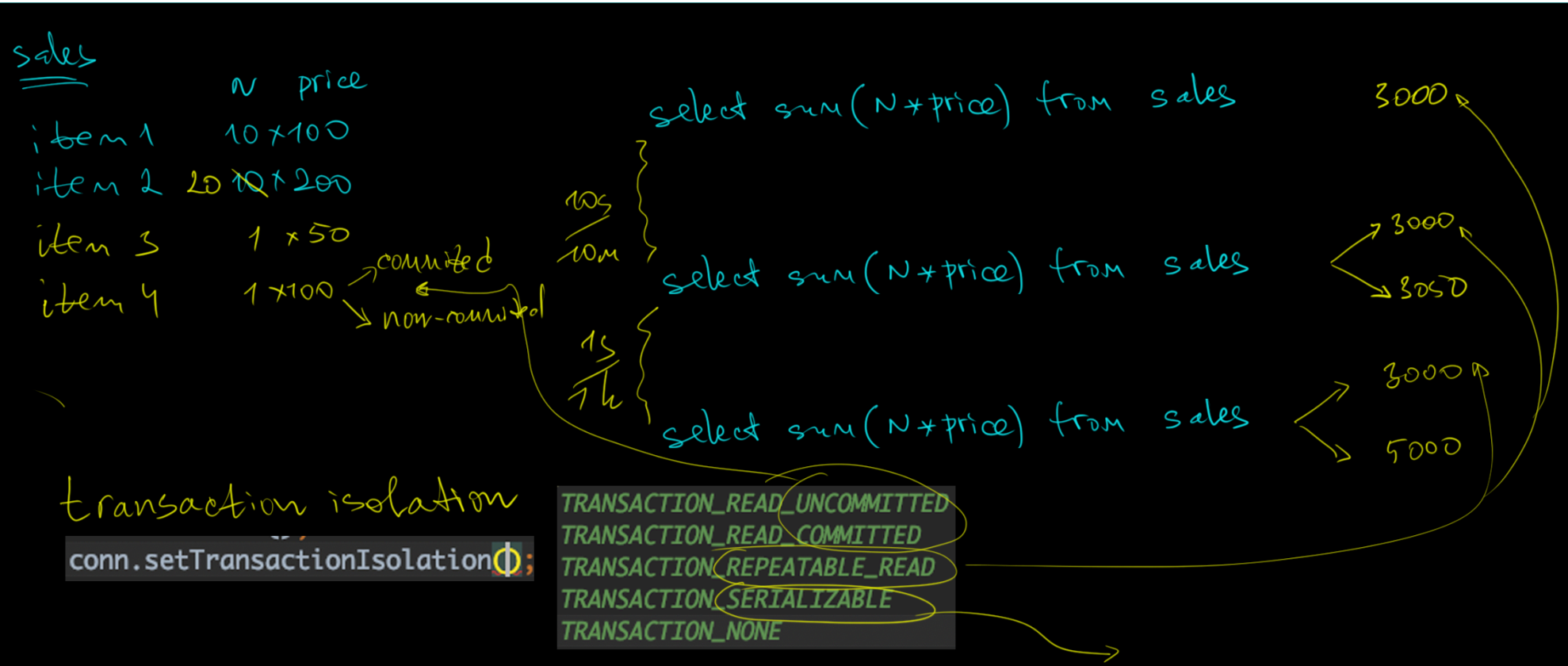
```
conn.setAutoCommit(false);
```

data will be sent to the server
BUT, will be persisted only after `conn.commit();`
OR, can be cancelled by `conn.rollback();`

You can guarantee that set of statements are done in batch

- ALL
- NOTHING





transaction isolation

```
conn.setTransactionIsolation();
```

```
TRANSACTION_READ_UNCOMMITTED
TRANSACTION_READ_COMMITTED
TRANSACTION_REPEATABLE_READ
TRANSACTION_SERIALIZABLE
TRANSACTION_NONE
```

select sum(N*price) from sales
select sum(N*price) from sales <time