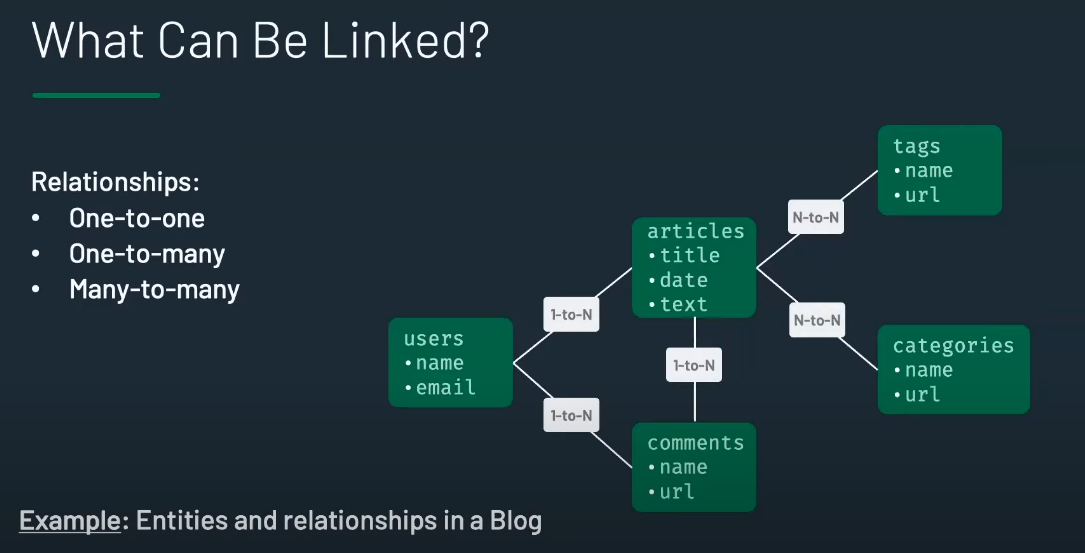
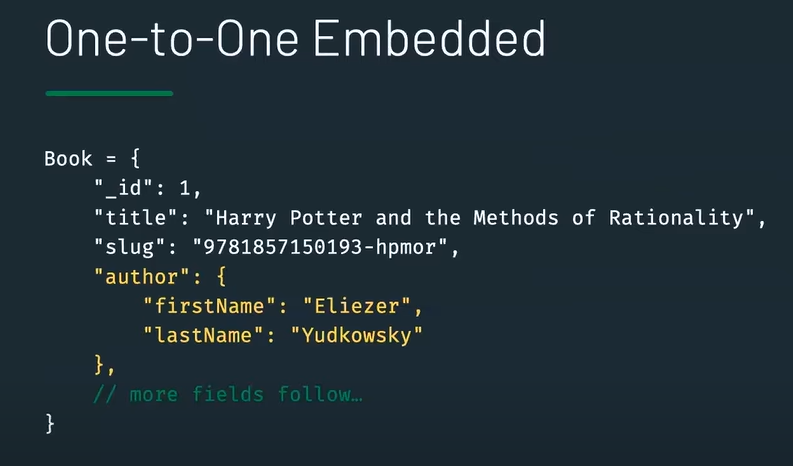
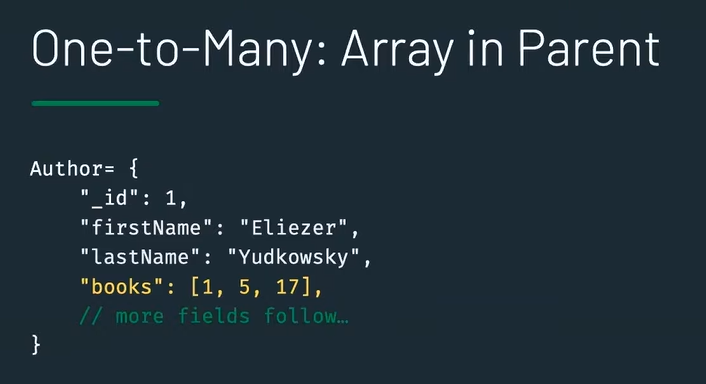
Table – collection

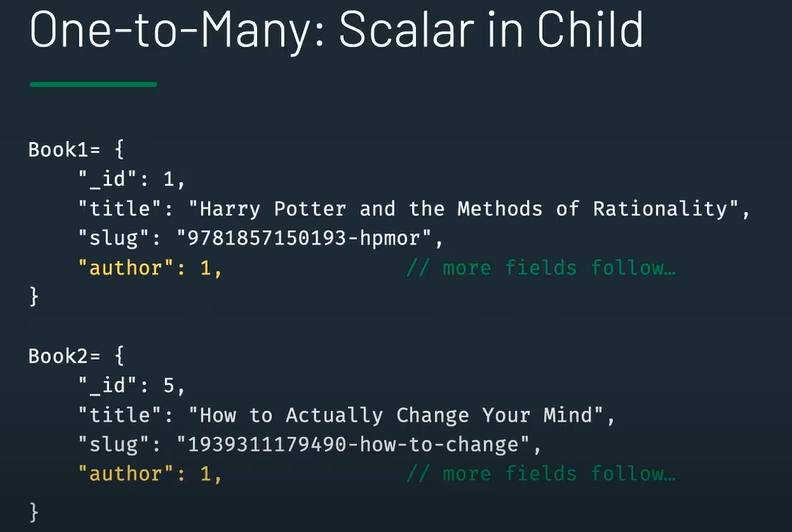
Rows – documents

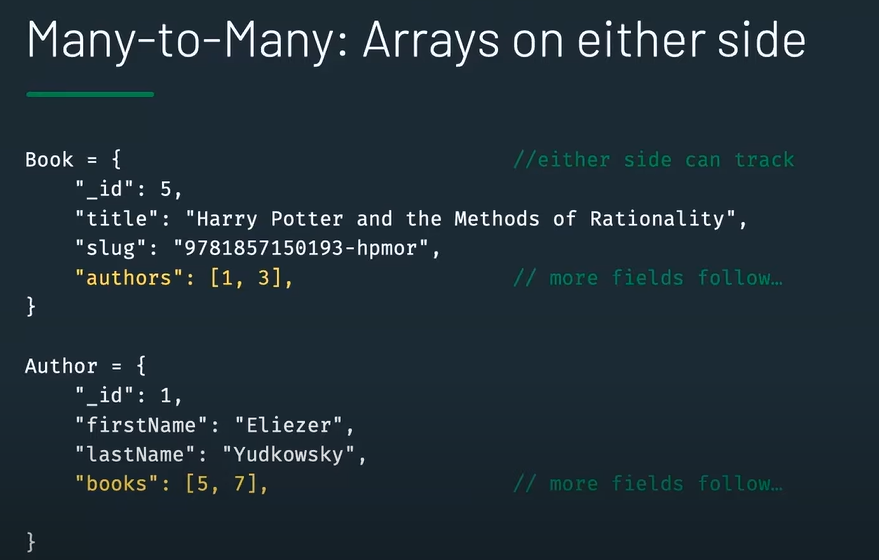


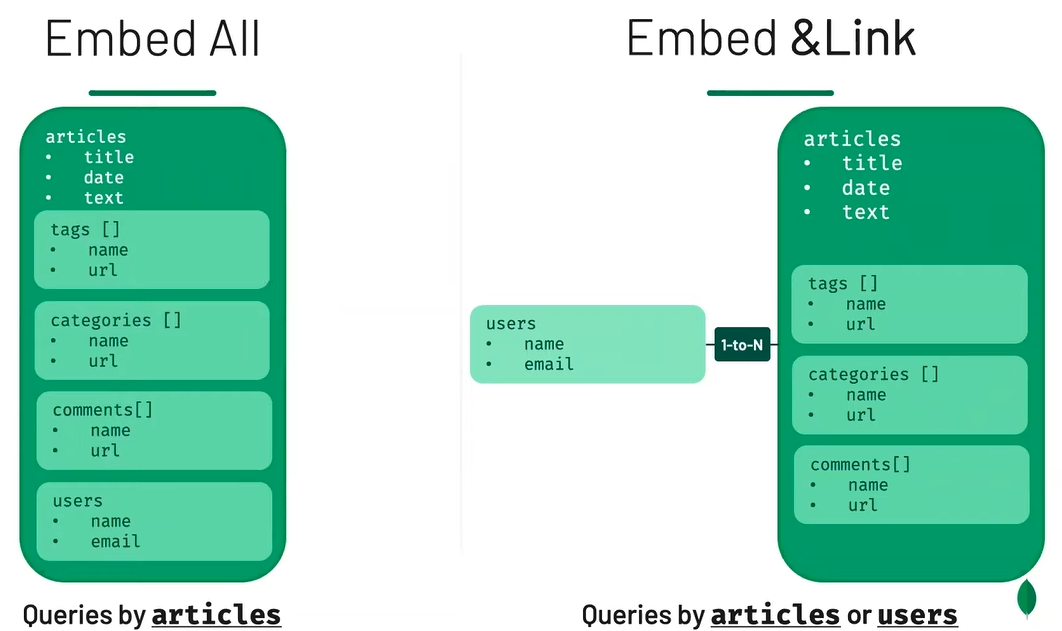


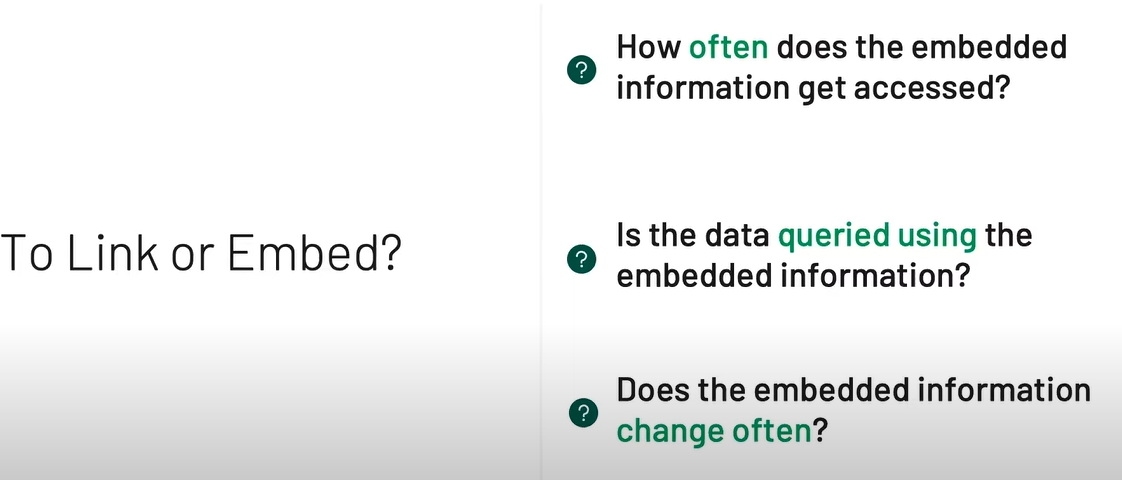


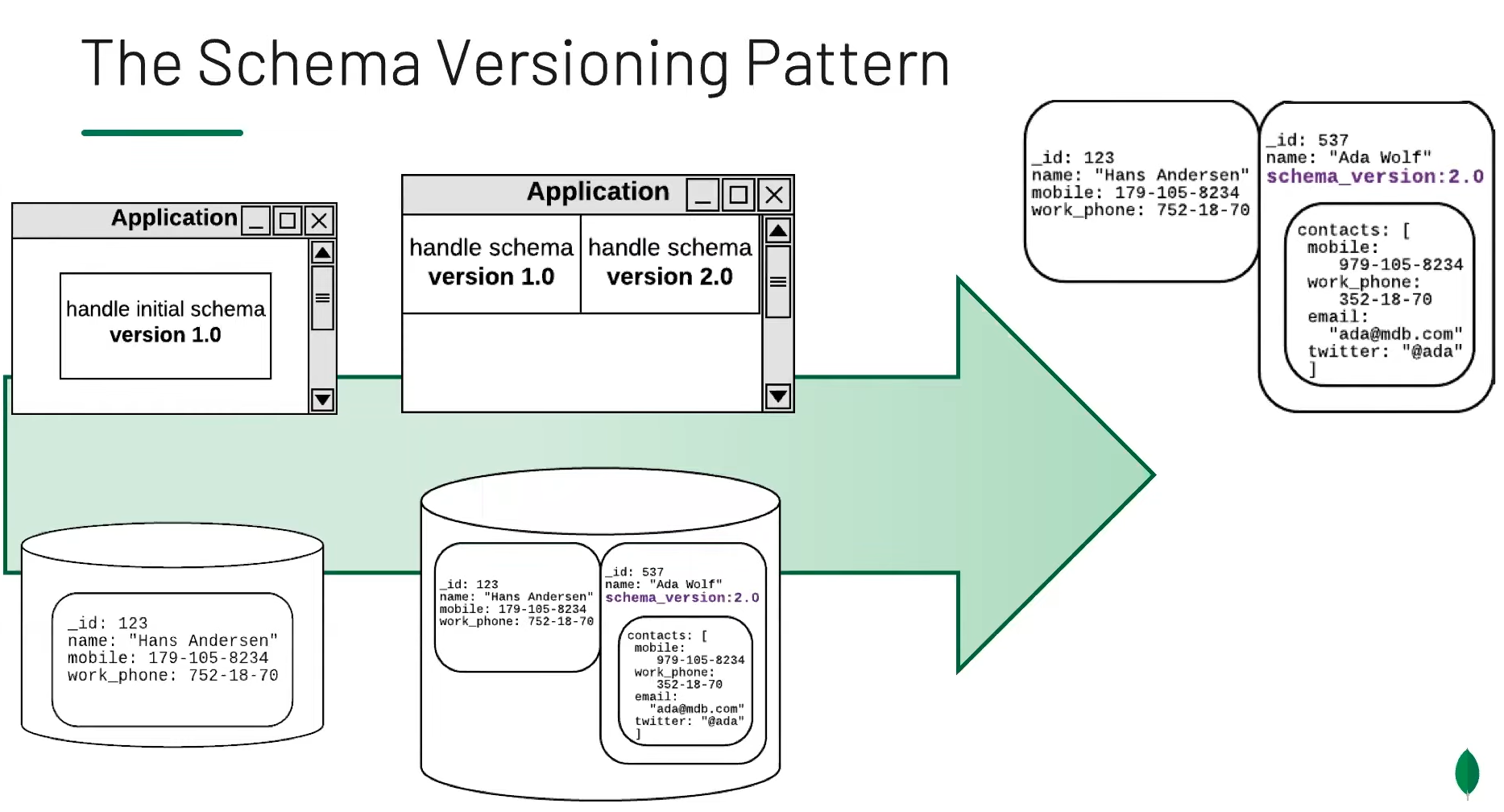


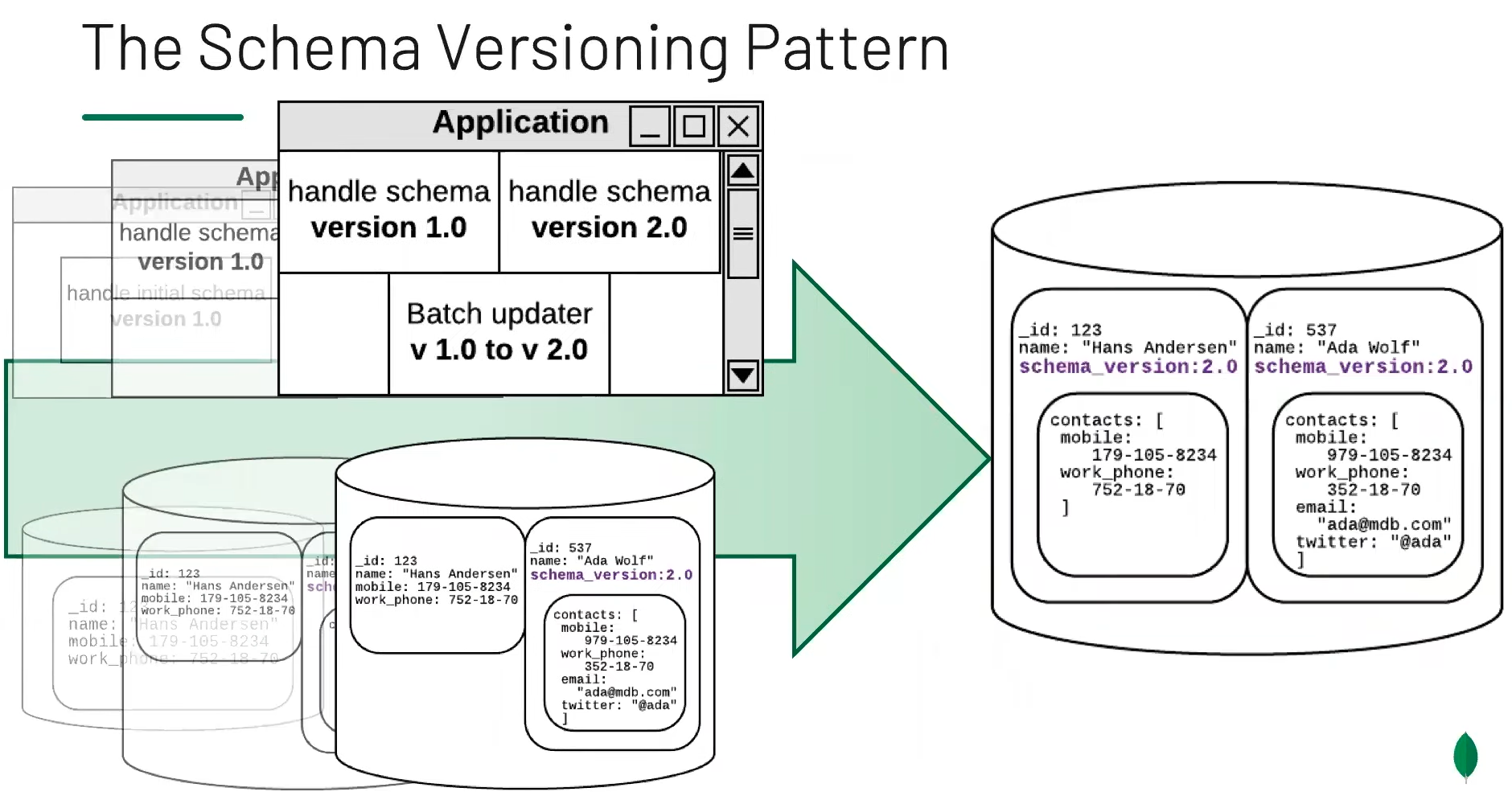


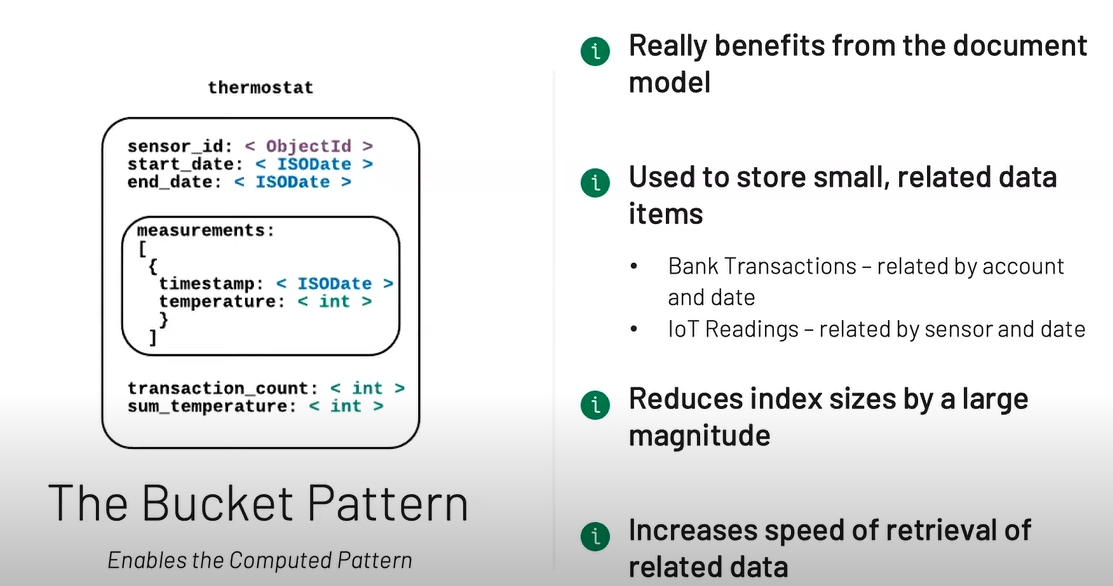


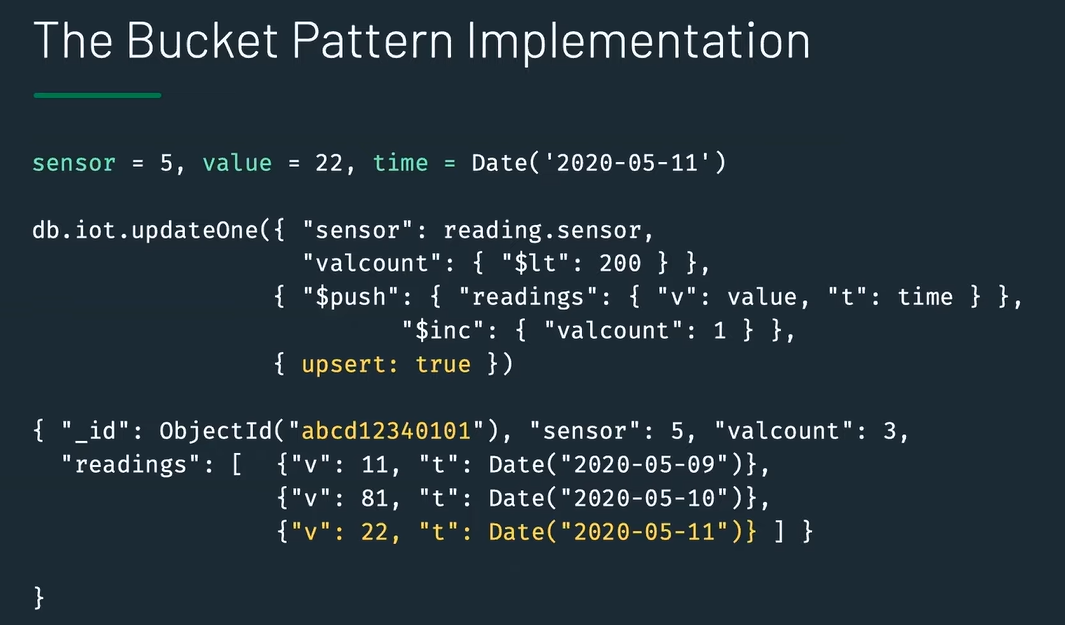




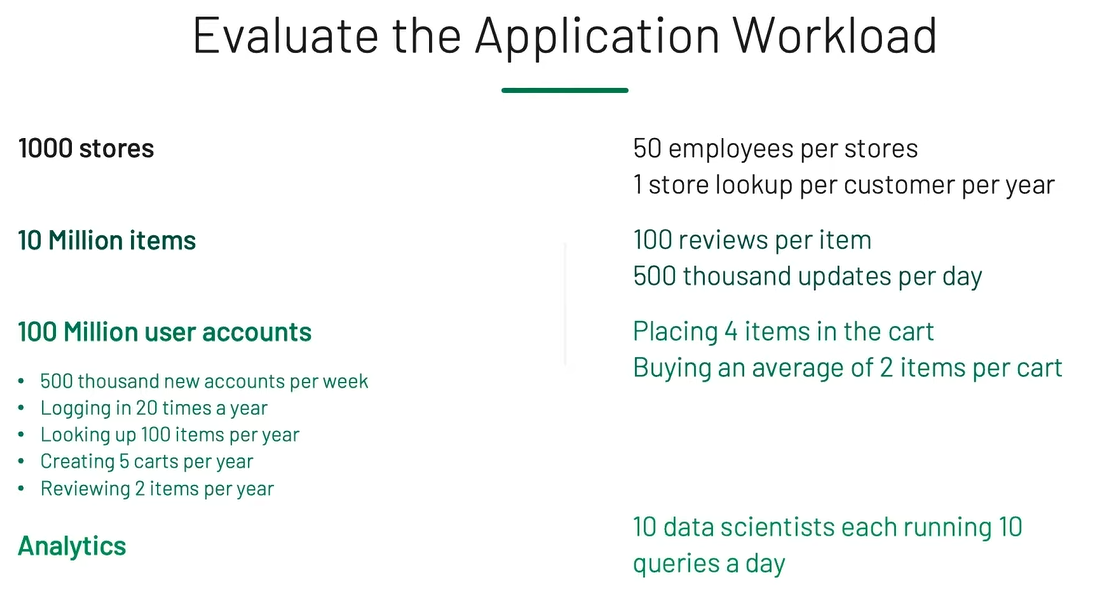


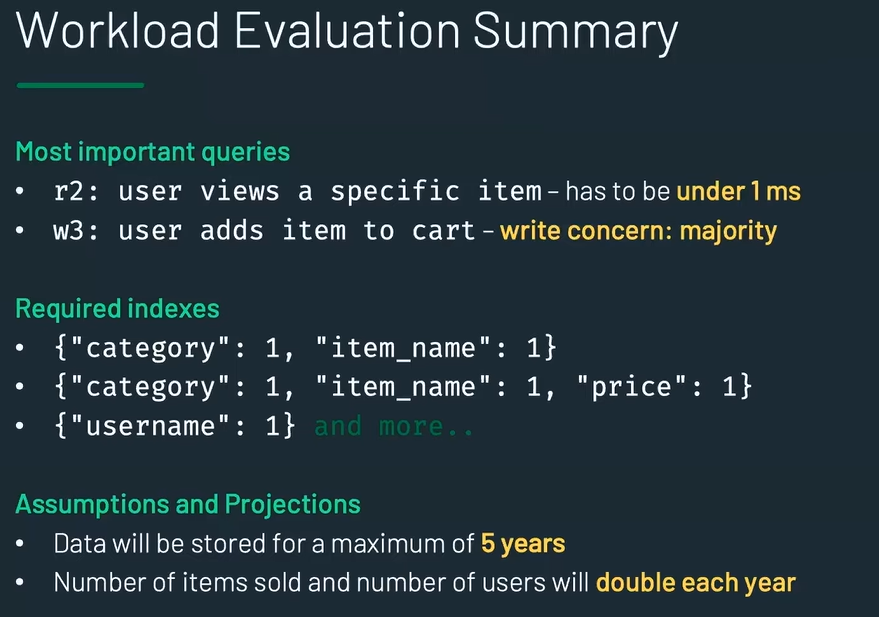




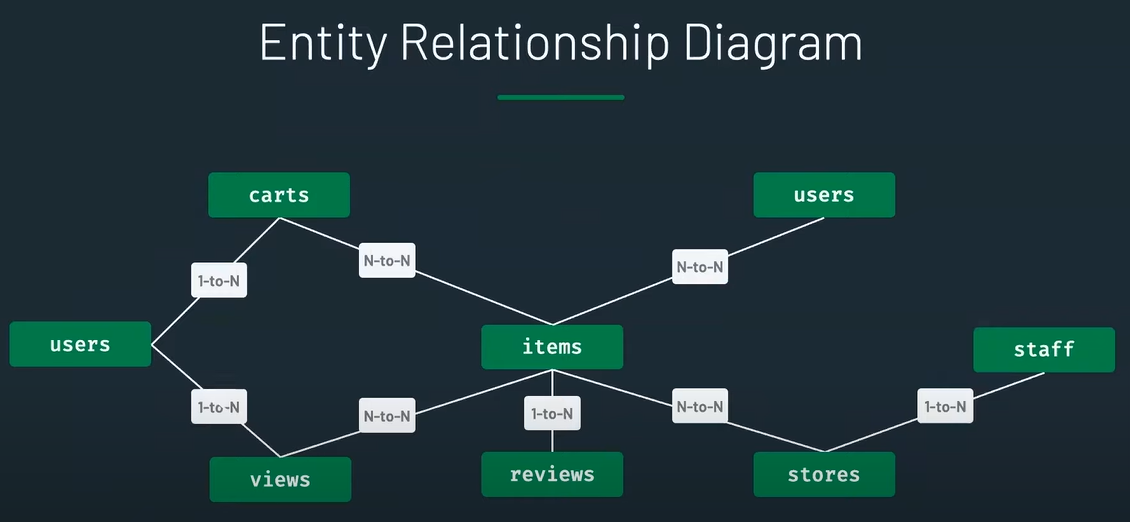


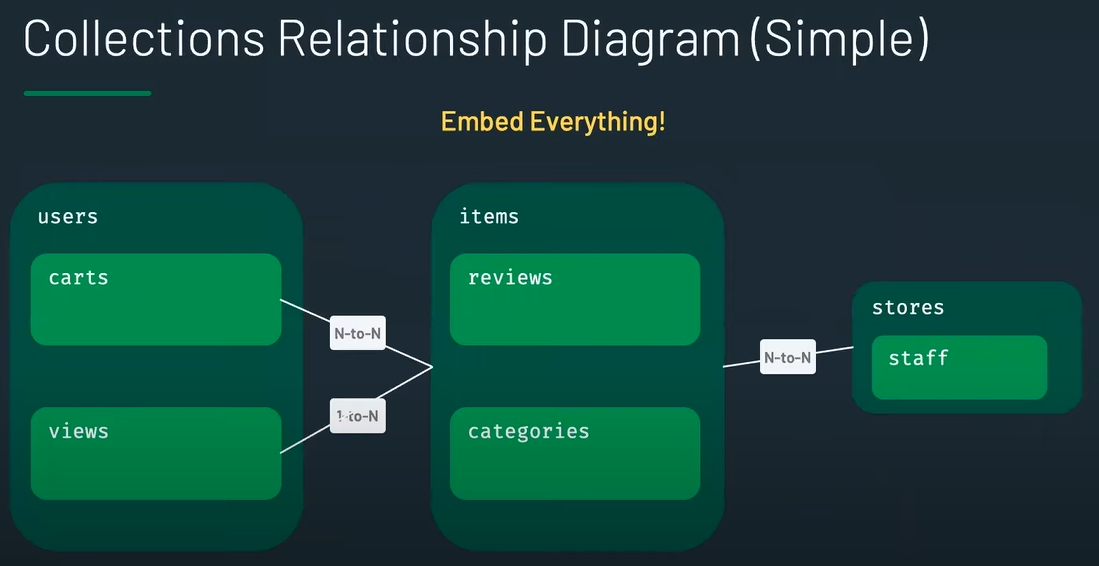


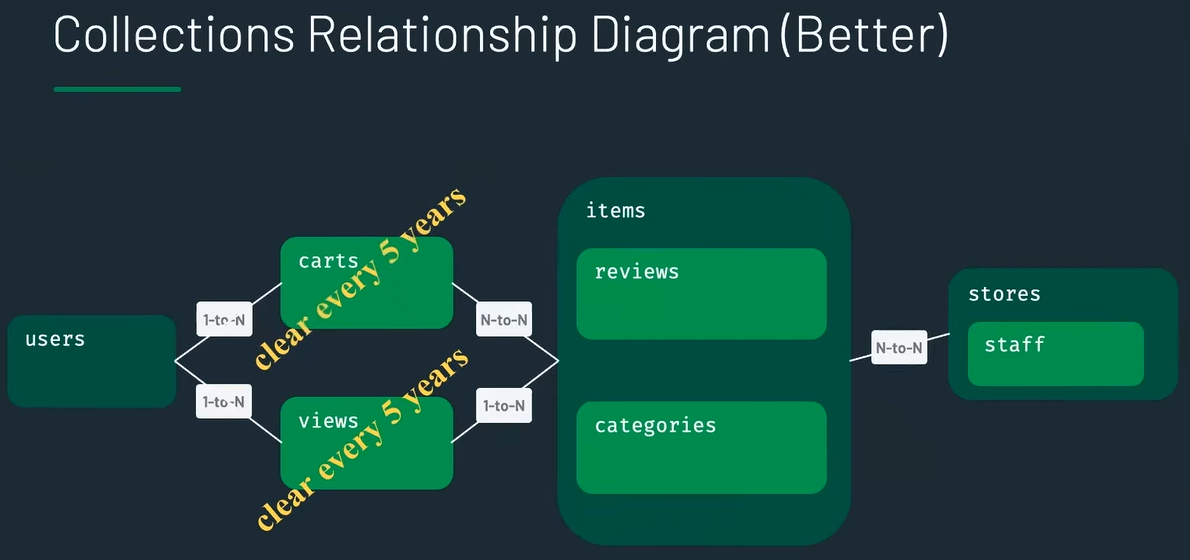


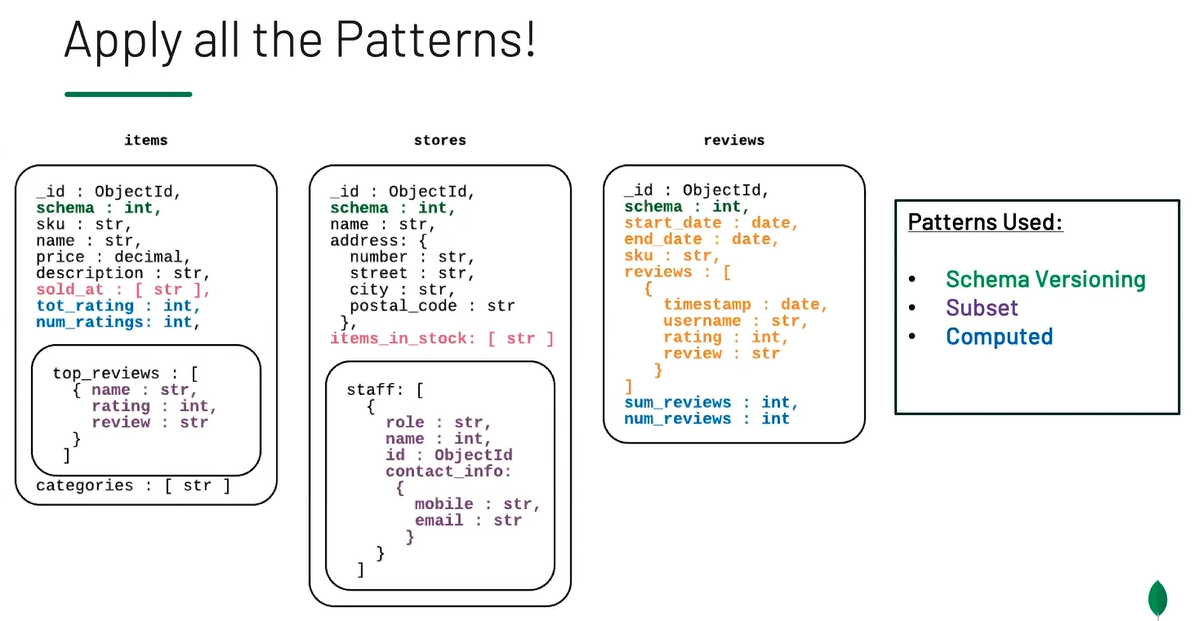


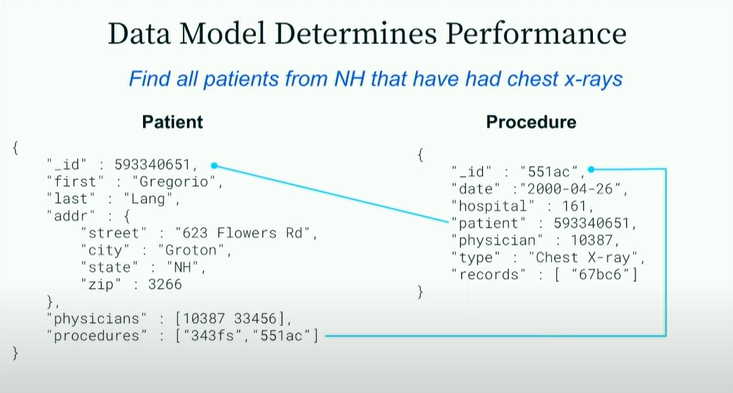


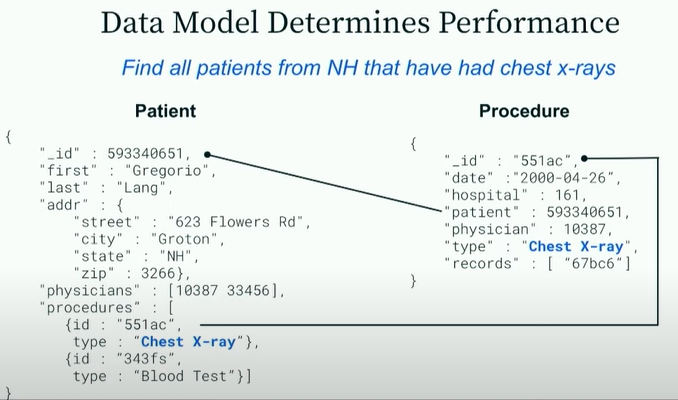








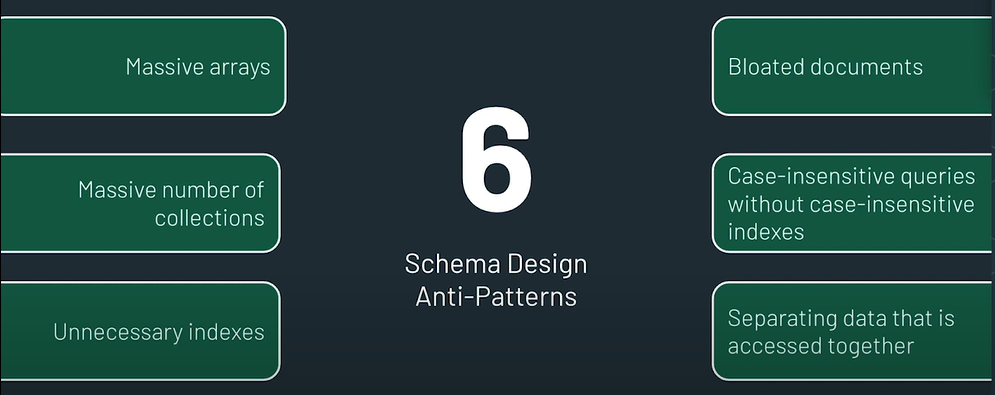




Yes, type filed is duplicated, but

This is better because the name “Chest X-ray” never changes in medical field. But if it does we need to update not only procedure document but also patient document

Mongo DB design anti patterns



Create db

use <db\_name>

Create collection

db.createCollection(“students”)

db.createCollection(“students” , {capped : true , size : 10000000 , max : 100} , autoIndexId : false} )

capped – tells there is a limit for collection size

size – limits the size of collection

max – limits the no. of documents

autoIndexId – tells to create automatic id index

Drop db

db.dropDatabase()

insert

db.insertOne({ name : “shakthi” ,

gpa : 2.8,

graduationDate : null,

registeredDate : new Date(),

courses : [“bio” , “chem” , “maths”],  
address : {street : “Madurai Veeran St.”,  
 city : “chennai”,  
 zip : 600101}  
})

db.insertMany([{ name : “shakthi” ,

gpa : 2.8 ….},

{ name : “guna” ,

gpa : 4.3 ….},

{ name : “maran” ,

gpa : 2.6 ….}

])

db.find().sort({name : 1}) - sorts according to ascending order

db.car.find().skip(5).limit(5)

skip – starts taking data after ignoring first 5 rows;

limit – returns only 5 rows

In this case, returns rows from 6-10

find() takes 2 argument , 1st – query , 2nd – projection

find({query} , {projection})

db.find({} , {\_id: false , name : true})



db.find({name : {$ne : “shakthi”} })

db.find({age : {$gt : 22} })

db.find({age : {$lt : 22} })

db.find({age : {$gte : 22} })

db.find({age : {$gte : 22 , $lte : 30} })

db.find({name : {$in : [“shakthi” , “guna”]} })

db.find({name : {$nin : [“shakthi” , “guna”]} })

($and , $or , $not , $nor)

db.find({$and : [{fullTime : true} , {age : {$lte : 22}}] })

db.find({name : {$in : [“shakthi” , “guna”]} })

db.find({age : {$gte : 22} })

if a document does not have a value or is set to null those fields will be include in the above query

db.find({age : {$not : {$gte : 22}} })

this query will return all documents with age > 22 and also the ones whose age : null

show dbs – list all DB

show collections – list all collections inside that particular DB

quit() –quit

mongo --host localhost --port 27017 -u <username> -p <password> --authenticationDatabase <authDB> <dbname>

mongo --host localhost --port 27017 -u myUserAdmin -p myPassword --authenticationDatabase admin testDB

- connect to DB {

--host = hostname

--port = port

-u = username  
 -p = password

testDB is the DB that I wanna connect to

}

load("/Users/shakthi/Downloads") - execute the contents in the file {

/Users/shakthi/Downloads - filepath

}

db.collection.find().pretty()

Table = person

db.person.find({

date\_of\_birth: {

$gte: ISODate('1990-04-01T00:00:00.000Z'),

$lte: ISODate('2004-08-29T00:00:00.000Z')

}

})

db.person.find({

country\_of\_birth: { $in: ['FRANCE', 'ITALY', 'GERMANY'] }

})

db.person.find({

email\_id: /.\*\.com$/

}) {

Gets all records with email ending in “.com”

}

db.person.find({

email\_id: /@google/

}) {

Gets all records with email having @google in them

}

Select country\_of\_birth , count(\*) FROM person GROUP BY country\_of\_birth HAVING COUNT(\*) >= 40 ORDER BY country\_of\_birth;

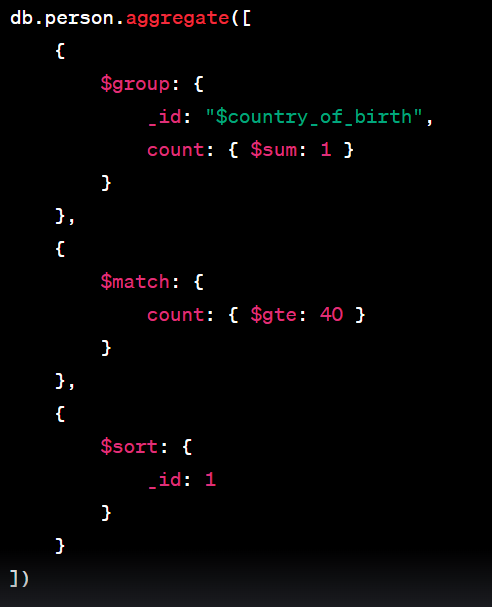


Table = car

Select MAX(price) FROM car;

db.car.aggregate([

{

$group: {

\_id: null,

max\_price: { $max: "$price" }

}

}

])

Select AVG(price) FROM car;

db.car.aggregate([

{

$group: {

\_id: null,

avg\_price: { $avg: "$price" }

}

}

])

Select ROUND(AVG(price)) FROM car;

db.car.aggregate([

{

$group: {

\_id: null,

avg\_price: { $avg: "$price" }

}

},

{

$project: {

rounded\_avg\_price: { $round: ["$avg\_price", 0] }

}

}

])

Select SUM(price) FROM car;

db.car.aggregate([

{

$group: {

\_id: null,

total\_price: { $sum: "$price" }

}

}

])

Select id,make,model, price AS original\_price , ROUND(price \* .10 , 2) AS ten\_percent , ROUND(price – (price \*.10) , 2) AS discount\_after\_10\_percent\_from\_car;

db.car.aggregate([

{

$project: {

id: 1,

make: 1,

model: 1,

original\_price: "$price",

ten\_percent: { $round: { $multiply: ["$price", 0.10] } },

discount\_after\_10\_percent\_from\_car: { $round: { $subtract: ["$price", { $multiply: ["$price", 0.10] }] } }

}

}

])

NULLIF returns null, if argument 1 = argument 2, else it returns argument 1

SELECT NULLIF(0 , 0) - null

SELECT NULLIF(10 , 10) - null

SELECT NULLIF(10 , 1) - 10

SELECT 10 /0 - ERROR can not divide by 0

SELECT 10/NULL - NULL

SELECT 10/NULLIF(2 , 9) - 5

SELECT 10/NULLIF(0 , 0) - null

SELECT COALESCE(NULL , 8) - 8

SELECT COALESCE(10,NULL,8) - 10

SELECT COALESCE(10, 8) - 10

SELECT COALESCE(NULLIF(0,0) , 7) - 7

SELECT NOW(); - gives timestamp

now

----------------------------------

2022-08-15 16:40:57.894242+05:30

SELECT NOW()::DATE;

db.runCommand({ serverStatus: 1 }).localTime - gives DATE

ISODate("2023-11-01T06:32:32.372Z")

SELECT NOW()::TIME; - gives TIME

db.collection.aggregate([

{

$project: {

time: {

$dateToString: {

format: "%H:%M:%S", // Format for time

date: new Date() // Current date

}

}

}

}

])

postgres=# SELECT NOW() - interval '10 years';

?column?

----------------------------------

2012-08-15 18:33:59.528471+05:30

postgres=# SELECT NOW() + INTERVAL '10 years';

? column?

----------------------------------

2032-08-15 18:35:54.356071+05:30

postgres=# SELECT (NOW() + INTERVAL '10 years')::DATE;

date

------------

2032-08-15

postgres=# SELECT EXTRACT(YEAR FROM NOW());

extract

---------

2022

postgres=# SELECT EXTRACT(MONTH FROM NOW());

extract

---------

8

postgres=# SELECT AGE(NOW() , '2001-04-22'::DATE);

age

-----------------------------------------

21 years 3 mons 23 days 18:48:10.681282

Primary key constraint

ALTER TABLE person ADD PRIMARY KEY (id);

ALTER TABLE person DROP CONSTRAINT person\_pkey;

Unique constraint

ALTER TABLE person ADD CONSTRAINT unique\_email\_address UNIQUE (email);

ALTER TABLE person DROP CONSTRAINT unique\_email\_address;

Check constraint

ALTER TABLE person ADD CONSTRAINT gender\_constraint CHECK (gender = ‘FEMALE’ OR gender = ‘MALE’);

AUTO\_INCREMENT (use constraint near a column during CREATE)

DEFAULT value (use constraint near a column during CREATE)

ALTER TABLE table\_name DROP CONSTRAINT some\_name;

DELETE FROM person ; deletes all rows

db.students.deleteMany({})

DELETE FROM person WHERE gender = ‘FEMALE’;

db.students.deleteMany({fullTime : false})

UPDATE person SET email = ‘omar@gmail.com’ WHERE name = ‘omar’;

db.students.updateOne({name : “shakthi”} , {${set} : {gpa : 4} })

db.students.updateOne({\_id : ObjectId(“sdf76sdf87t8f83bshdf82”)} , {${set} : {gpa : 4} })

db.students.updateOne({\_id : ObjectId(“sdf76sdf87t8f83bshdf82”)} , {${unset} : {gpa : “”} })

removes the field gpa from that document

db.students.updateMany({} , {${set} : {fullTime : false} })

sets all documents’ fullTime field to false  
if they do not have fullTime field, one will be created for them

db.students.updateMany({fullTime : {$exists : false}} , {${set} : {fullTime : true} })

sets fullTime field to true for all the documents who do not have fullTime field

UPSERT(update + insert)

ON CONFLICT DO - to avoid error when PK is already present or UNIQUE constraint is violated

* To insert a new row. If a row already exists with given values it updates it

Record with id =1 exists already in the table

1. INSERT INTO person(id ,name , email , date\_of\_birth , country\_of\_birth) VALUES(1 , ‘RUSS’ , ‘MALE’ , ‘rr@gmail.com’ , DATE ‘1995-04-22’ , ‘Norway’) ON CONFLICT DO NOTHING;

Output : INSERT

1. 0
2. INSERT INTO person(id ,name , email , date\_of\_birth , country\_of\_birth) VALUES(1 , ‘RUSS’ , ‘MALE’ , ‘rr@gmail.com’ , DATE ‘1995-04-22’ , ‘Norway’) ON CONFLICT DO UPDATE SET email = EXCLUDED.email , name = EXCLUDED.name ;

Output : INSERT

1. 1

(name and email are updated for the record with id=1)

FOREIGN KEY

CREATE TABLE person(

id BIGSERIAL NOT NULL PRIMARY KEY,

name VARCHAR(50) NOT NULL,

gender VARCHAR(7),

email VARCHAR(100),

date\_of\_birth DATE NOT NULL,

country\_of\_birth VARCHAR(20) NOT NULL,

car\_id BIGINT REFERENCES car(id),

UNQUE(car\_id)

);

CREATE TABLE car(

id BIGSERIAL NOT NULL PRIMARY KEY,

make VARCHAR(100) NOT NULL,

model VARCHAR(100) NOT NULL,

price NUMERIC(19,2) NOT NULL

);

Table with 2 primary keys

Table having a column which is both PRIMARY and FOREIGN key

CREATE TABLE branch\_supplier(

branch\_id INT ,

supplier\_name VARCHAR(40),

suplply\_type VARCHAR(40),

PRIMARY KEY(branch\_id , supplier\_name),

FOREIGN KEY(branch\_id) REFERENCES branch(id) ON DELETE CASCADE

);

ON DELETE CASCADE – deletes the rows in branch\_supplier table when that particular id is deleted in branch table

ON DELETE SET NULL – sets the branch\_id column value to NULL when that particular id is deleted in branch table

JOINS

INNER JOIN

SELECT \* FROM person JOIN car ON person.car\_id = car.id;

LEFT JOIN

SELECT \* FROM person LEFT JOIN car ON person.car\_id = car.id;

Gives all records from PERSON combined with empty records on CAR table

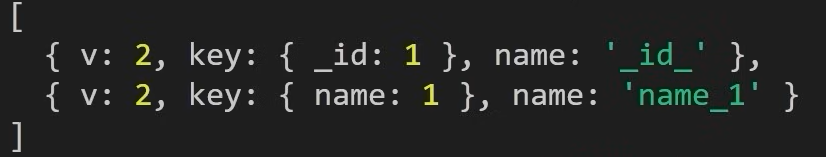
SELECT \* FROM person LEFT JOIN car ON person.car\_id = car.id WHERE car.\* IS NULL;

car.\* IS NULL = ( car.id IS NULL AND car.make IS NULL AND car.model IS NULL car.price IS NULL )

Indexes

db.students.createIndex({ name : 1}) indexes in asc order

db.students.getIndexes()



db.dropIndex(“name\_1”)

UNIONS

UNION Operator

used to combine the results of two or more SELECT statements without returning any duplicate rows.

testdb=# SELECT EMP\_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT

ON COMPANY.ID = DEPARTMENT.EMP\_ID

UNION

SELECT EMP\_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT

ON COMPANY.ID = DEPARTMENT.EMP\_ID;

UNION ALL Operator

Same as UNION but also returns duplicate rows

testdb=# SELECT EMP\_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT

ON COMPANY.ID = DEPARTMENT.EMP\_ID

UNION ALL

SELECT EMP\_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT

ON COMPANY.ID = DEPARTMENT.EMP\_ID;