Fundamentals of relational database systems

Katharina Rasch

PyConDE / PyData Berlin 2022

ABOUT

Dr. Katharina Rasch

Data scientist | computer vision engineer | teacher

PhD Computer Science (KTH Stockholm)

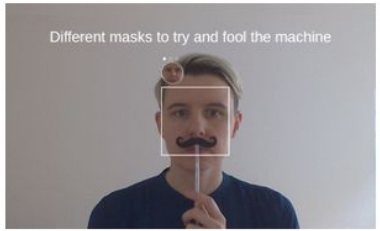
Freelancer in Berlin → Work with me

hello@krasch.io | github | twitter

https://krasch.io

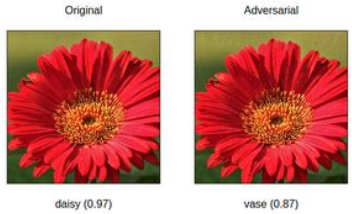
kat@krasch.io

PROJECTS



The face recognition game

Helping people explore the breaking points of a face recognition system



Generating adversarial images in the browser using tensorflow.js

Live demo + full code + details on memory management / async execution

OVER 100 Data Scientist Interview Questions and Answers!

Interview Questions from Amazon, Google, Facebook, Microsoft, and more!

I know this is long...

Really long. But don't be intimidated by the length — I have broken this down into four sections (machine learning, stats, SQL, miscellaneous) so that you can go through this bit by bit.

https://towardsdatascience.com/over-100-data-scientist-interview-questions-and-answers-c5a66186769a

Technical interview questions

⚠ The answers here are given by the community. Be careful
If you see an error, please create a PR with a fix

The list is based on this post

Table of contents

- SQL
- Coding (Python)
- Algorithmic Questions

https://github.com/alexeygrigorev/data-science-interviews/blob/master/technical.md

Preparing For Data Science Interview? Here is a Complete Guide To Help You Perform Well

About best approaches, strategies, and tips to get hired

[...]

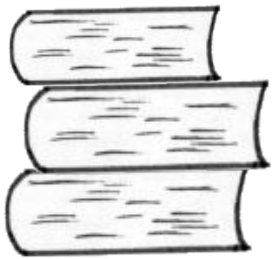
SQL

LearnSQL is a platform that exclusively focuses on building and testing SQL skills. They have many free scenario-based coding challenges. These will be very useful to learn SQL. Learning SQL has always been very tricky. Because to learn real-life SQL skills you need to have access to a real-world database. These platforms make the scenarios real and very well replicate the real-life problems. Thus creating a wonderful learning experience.

https://towardsdatascience.com/preparing-for-data-science-interview-here-is-a-complete-guide-to-help-you-perform-well-a98d28f4a1f4

let’s have a look at

what’s behind SQL



versus

1. Relational model / algebra

2. Query planner and indices

1. Relational model / algebra

2. Query planner and indices

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

Relational model:

all data is expressed as tables (relations) and relationships* between the tables

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

* unfortunate that “relationships” also contains “relation”, but that is not what is meant here

Relational algebra:
operations on tables

Relational algebra: operations on tables

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...

Relational algebra: operations on tables

Projection Π: Grab only the columns you need

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...

Relational algebra: operations on tables

Projection Π: Grab only the columns you need

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...

```
SELECT name, description
FROM product;
```

Relational algebra: operations on tables

Selection σ: Grab only the rows you need

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...

```
SELECT *
FROM product
WHERE category_id = 2;
```

Relational algebra: operations on tables

Join ⋈: Combine two tables

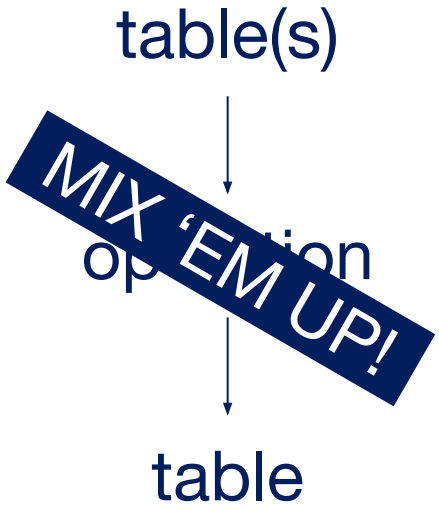
id	name	category_name
1	bread	food
2	apple	food
3	jeans	clothing

```
SELECT product.id,
       product.name
       category.name AS category_name,
FROM product
JOIN category
ON product.category_id = category.id;
```

Relational algebra:
operations* on tables

* there are a few additional operations, let's skip those today

Relational algebra:
operations* on tables



* there are a few additional operations, let's skip those today

Complex SQL query I can't figure out

Asked 7 years, 9 months ago Modified 7 years, 8 months ago Viewed 1k times

-3

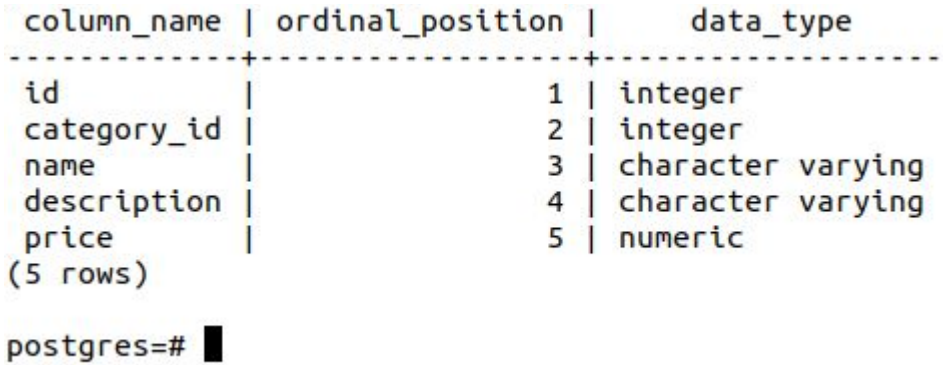
[...]

```
select
    c.covered, t.total
from
    (SELECT DISTINCT
        count(distinct unique_id) as covered
    FROM
        requirement
    WHERE
        requirement.id IN (select
                requirement.id
            from
                `requirement` requirement
            INNER JOIN `step_has_requirement` step_has_requirement ON requirement.
            INNER JOIN `step` step ON step_has_requirement.`step_id` = step.`id`
                AND step.`test_case_id` = step_has_requirement.`step_test_case_id`
                AND step.`test_case_test_id` = step_has_requirement.`step_test_cas
            INNER JOIN `test_case` test_case ON step.`test_case_id` = test_case.`i
                AND test_case.`test_id` = step.`test_case_test_id`
            INNER JOIN `test` test ON test_case.`test_id` = test.`id`
            INNER JOIN `test_plan_has_test` test_plan_has_test ON test.`id` = test
            INNER JOIN `test_plan` test_plan ON test_plan_has_test.`test_plan_id`
                AND test_plan.`test_project_id` = test_plan_has_test.`test_plan_te
            INNER JOIN `test_project` test_project ON test_plan.`test_project_id`
            INNER JOIN `requirement_status` requirement_status ON requirement.`rec
            INNER JOIN `requirement_spec_node` requirement_spec_node ON requiremer
                AND requirement_spec_node.`requirement_spec_project_id` = requirem
                AND requirement_spec_node.`requirement_spec_spec_level_id` = requi
                AND requirement_spec_node.`requirement_spec_id` = requirement.`rec
            INNER JOIN `requirement_spec` requirement_spec ON requirement_spec_noc
                AND requirement_spec.`project_id` = requirement_spec_node.`require
                AND requirement_spec.`spec_level_id` = requirement_spec_node.`requ
            INNER JOIN `project` project ON requirement_spec.`project_id` = projec
            WHERE
                requirement_status.status = 'general.approved'
```

https://stackoverflow.com/questions/24516369/complex-sql-query-i-cant-figure-out

even the table metadata is in a table!

```
SELECT column_name,
       ordinal_position,
       data_type
FROM information_schema.columns
WHERE table_name = 'product'*;
```



```
 column_name | ordinal_position |     data_type
-------------+------------------+-------------------
 id          |                1 | integer
 category_id |                2 | integer
 name        |                3 | character varying
 description |                4 | character varying
 price       |                5 | numeric
(5 rows)

postgres=#
```

* example specific to PostgreSQL

it’s tables

all the

way down

so what?

data independence

1. Relational model / algebra

2. Query planner and indices

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

Selection σ

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

Projection Π

Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

Selection σ

order of the operations?

let’s try it out!

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

1. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

id	category_id	name	description	price	id’	name’	icon
1	1	bread	...	...	1	food	...
2	1	apple	...	...	1	food	...
3	2	jeans	...	...	2	clothing	...
...	...	...	...	...	...	...	...
10000	...	...	...	...	...	...	....

1. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

id	category_id	name	description	price	id’	name’	icon
1	1	bread	...	...	1	food	...
2	1	apple	...	...	1	food	...
3	2	jeans	...	...	2	clothing	...
...	...	...	...	...	...	...	...
10000	...	...	...	...	...	...	....

1. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

2. Selection σ

id	category_id	name	description	price	id’	name’	icon
1	1	bread	...	...	1	food	...
2	1	apple	...	...	1	food	...
3	2	jeans	...	...	2	clothing	...
...	...	...	...	...	...	...	...
10000	...	...	...	...	...	...	....

1. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

2. Selection σ

id	category_id	name	description	price	id'	name'	icon
1	1	bread	...	...	1	food	...
2	1	apple	...	...	1	food	...
3	2	jeans	...	...	2	clothing	...
...	...	...	...	...	...	...	...
10000	...	...	...	...	...	...	....

3. Projection Π

1. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

2. Selection σ

id	category_id	name	description	price	id'	name'	icon
1	1	bread	...	...	1	food	...
2	1	apple	...	...	1	food	...
3	2	jeans	...	...	2	clothing	...
...	...	...	...	...	...	...	...
10000	...	...	...	...	...	...	....

3. Projection Π

1. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

2. Selection σ

not very efficient...

let’s try again!

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

1. Selection σ

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

1. Selection σ

product

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	...	...	...

category

id	name	icon
1	food	...
2	clothing	...
3	...	...

2. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

1. Selection σ

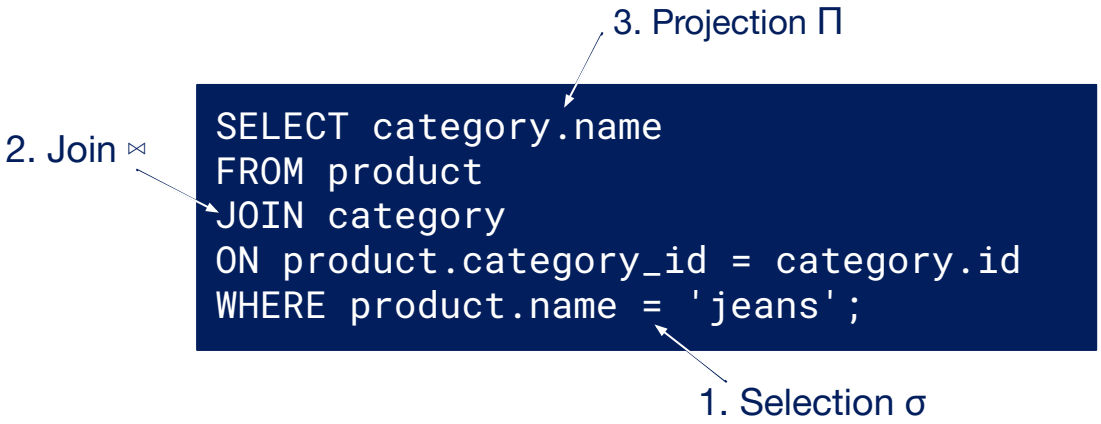
id	category_id	name	description	price	id’	name’	icon
3	2	jeans	...	...	2	clothing	...

2. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

1. Selection σ

id	category_id	name	description	price	id’	name’	icon
3	2	jeans	...	...	2	clothing	...



id	category_id	name	description	price	id'	name'	icon
3	2	jeans	...	...	2	clothing	...

3. Projection Π

2. Join ⋈

```
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

1. Selection σ

Query planner

devises execution plans

Plan 1: $\bowtie \rightarrow \sigma \rightarrow \Pi$

Plan 2: $\sigma \rightarrow \bowtie \rightarrow \Pi$

...

Plan n: ...

Query planner

devises execution plans

approximates their costs*

Plan 1: ⋈ → σ → Π $$$

Plan 2: σ → ⋈ → Π $

...

Plan n: ...

* using statistics such as number of rows, columns, distinct values, ...

Query planner

devises execution plans

approximates their costs

picks the best plan

Plan 1: ⋈ → σ → Π $$$

Plan 2: σ → ⋈ → Π $

...

Plan n: ...

do all database systems have a query planner?

```
EXPLAIN
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

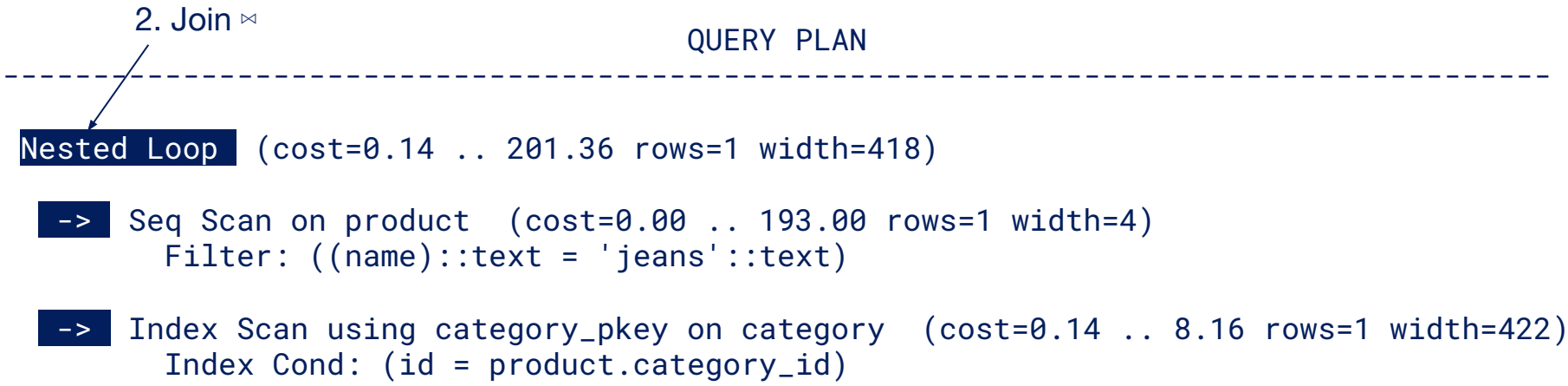
```
                                  QUERY PLAN
-------------------------------------------------------------------------------

 Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)

   ->  Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
         Filter: ((name)::text = 'jeans'::text)

   ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
         Index Cond: (id = product.category_id)
```

```
EXPLAIN
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

```
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)
   ->  Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
         Filter: ((name)::text = 'jeans'::text)
   ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
         Index Cond: (id = product.category_id)
```

1. Selection σ

```
EXPLAIN
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```



```
EXPLAIN
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

```
                                    QUERY PLAN
-------------------------------------------------------------------------------
                              $$$
 Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)

   ->  Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
         Filter: ((name)::text = 'jeans'::text)

   ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
         Index Cond: (id = product.category_id)
```

```
EXPLAIN
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

```
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
                                  $$$
 Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)
                                                  $$$
   ->  Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
         Filter: ((name)::text = 'jeans'::text)
                                                                  $
   ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
         Index Cond: (id = product.category_id)
```

```
EXPLAIN
SELECT category.name
FROM product
JOIN category
ON product.category_id = category.id
WHERE product.name = 'jeans';
```

```
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
                                  $$$
 Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)
                                                  $$$
   ->  Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
         Filter: ((name)::text = 'jeans'::text)
                                                                  $
   ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
         Index Cond: (id = product.category_id)
```

table scan

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	apple	...	...

table scan

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	apple	...	...

```
CREATE INDEX idx_product_name
ON product(name);
```

table scan

id	category_id	name	description	price
1	1	bread	...	...
2	1	apple	...	...
3	2	jeans	...	...
...	...	...	...	...
10000	...	apple	...	...

```
CREATE INDEX idx_product_name
ON product(name);
```

```
{'bread': [1],
 'apple': [2, 10000],
 'jeans': [3]
 ... } *
```

* in reality, index would typically be a tree structure, not a hashmap / dictionary

QUERY PLAN (WITHOUT INDEX)

```
Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)

  ->  Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
        Filter: ((name)::text = 'jeans'::text)

  ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
        Index Cond: (id = product.category_id)
```

QUERY PLAN (WITH INDEX)

```
Nested Loop  (cost=0.43 .. 16.66 rows=1 width=418)

  ->  Index Scan using idx_product_name on product  (cost=0.29 .. 8.30 rows=1 width=4)
        Index Cond: ((name)::text = 'jeans'::text)

  ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
        Index Cond: (id = product.category_id)
```

QUERY PLAN (WITHOUT INDEX)

```
Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)

  -> Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
       Filter: ((name)::text = 'jeans'::text)

  ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
       Index Cond: (id = product.category_id)
```

QUERY PLAN (WITH INDEX)

```
Nested Loop  (cost=0.43 .. 16.66 rows=1 width=418)

  -> Index Scan using idx_product_name on product  (cost=0.29 .. 8.30 rows=1 width=4)
       Index Cond: ((name)::text = 'jeans'::text)

  ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
       Index Cond: (id = product.category_id)
```

QUERY PLAN (WITHOUT INDEX)

```
Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)

  -> Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
       Filter: ((name)::text = 'jeans'::text)

  ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
       Index Cond: (id = product.category_id)
```

QUERY PLAN (WITH INDEX)

```
Nested Loop  (cost=0.43 .. 16.66 rows=1 width=418)

  -> Index Scan using idx_product_name on product  (cost=0.29 .. 8.30 rows=1 width=4)
       Index Cond: ((name)::text = 'jeans'::text)

  ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
       Index Cond: (id = product.category_id)
```

QUERY PLAN (WITHOUT INDEX)

```
Nested Loop  (cost=0.14 .. 201.36 rows=1 width=418)

  -> Seq Scan on product  (cost=0.00 .. 193.00 rows=1 width=4)
       Filter: ((name)::text = 'jeans'::text)

  ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
       Index Cond: (id = product.category_id)
```

QUERY PLAN (WITH INDEX)

```
Nested Loop  (cost=0.43 .. 16.66 rows=1 width=418)

  -> Index Scan using idx_product_name on product  (cost=0.29 .. 8.30 rows=1 width=4)
       Index Cond: ((name)::text = 'jeans'::text)

  ->  Index Scan using category_pkey on category  (cost=0.14 .. 8.16 rows=1 width=422)
       Index Cond: (id = product.category_id)
```

index for every column?

storage overhead

insertion overhead

relational model / algebra is a
beautifully simple but extremely powerful abstraction

data independence!

enables us to have cool things like the query planner

relational model / algebra is a
beautifully simple but extremely powerful abstraction

data independence!

enables us to have cool things like the query planner

kat@krasch.io

https://github.com/krasch/presentations