Performance Tips

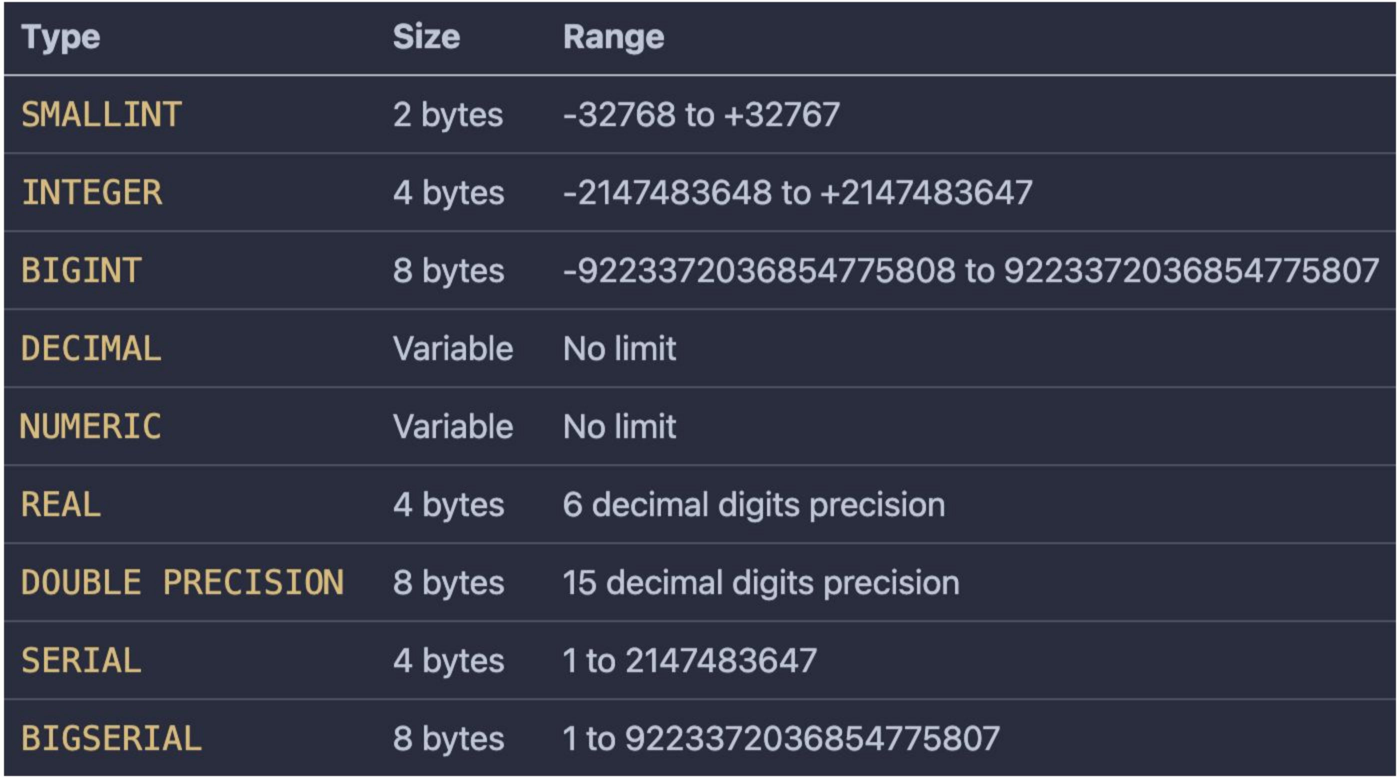
What is "performance"?

Performance: an action, task, or operation,
seen in terms of how successfully it was performed

- Oxford English Dictionary

Success in computing:	**Success in databasing:**
Correctness	Reliable SQL operations
Speed (Big O Notation)	Fast results
Memory Usage	Minimal memory usage

Data types



Type	Size	Range
SMALLINT	2 bytes	-32768 to +32767
INTEGER	4 bytes	-2147483648 to +2147483647
BIGINT	8 bytes	-9223372036854775808 to 9223372036854775807
DECIMAL	Variable	No limit
NUMERIC	Variable	No limit
REAL	4 bytes	6 decimal digits precision
DOUBLE PRECISION	8 bytes	15 decimal digits precision
SERIAL	4 bytes	1 to 2147483647
BIGSERIAL	8 bytes	1 to 9223372036854775807

Data modeling

The right database design is key to performance

Recall "normalization": eliminate redundant data

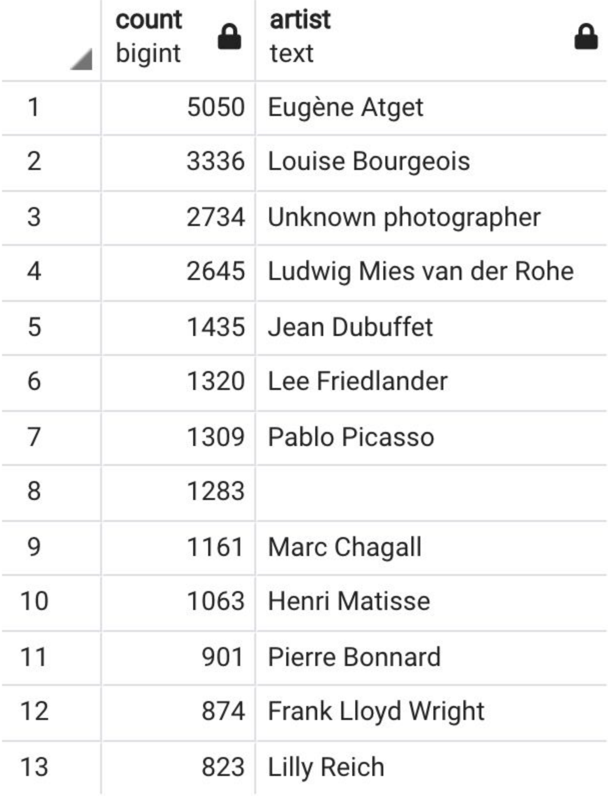
Counter example: moma_works table

```
SELECT count(artist) - count(DISTINCT artist)
FROM moma_works; -- 124466
```

124,466 redundant artist values

Data modeling

```
SELECT COUNT(*), artist
FROM moma_works
GROUP BY artist
ORDER BY count DESC;
```

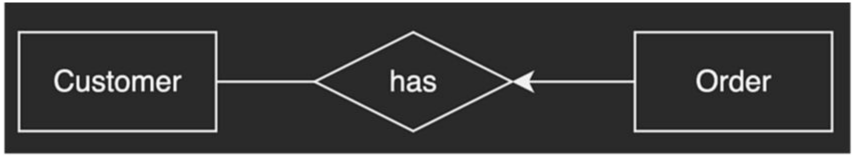


	count bigint	artist text
1	5050	Eugène Atget
2	3336	Louise Bourgeois
3	2734	Unknown photographer
4	2645	Ludwig Mies van der Rohe
5	1435	Jean Dubuffet
6	1320	Lee Friedlander
7	1309	Pablo Picasso
8	1283	
9	1161	Marc Chagall
10	1063	Henri Matisse
11	901	Pierre Bonnard
12	874	Frank Lloyd Wright
13	823	Lilly Reich

Data modeling

Tradeoff between speed (expensive JOINs)
and memory (wasted space)

Elegant queries



Task: Find orders belonging to customer #12

```
SELECT COUNT(*) > 0 FROM customers c JOIN orders o ON c.id = o.customer_id WHERE c.id = 12;
```

```
SELECT COUNT(*) > 0 FROM (SELECT 1 FROM orders WHERE customer_id = 12) as subq;
```

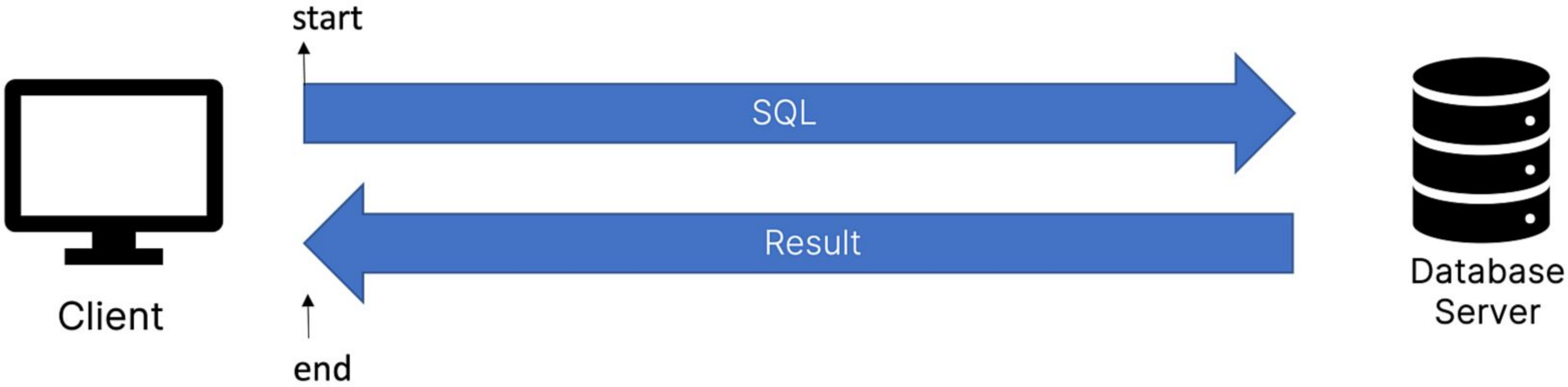
```
SELECT EXISTS(SELECT 1 FROM orders WHERE customer_id = 12);
```

Reduce round trips

SQL statements and results travel on network

Round-trip: full request-response cycle

Reduce round trips to improve efficiency



Reduce round trips: Example

```
INSERT INTO my_table (col1, col2, col3) VALUES (1, 2, 3);
INSERT INTO my_table (col1, col2, col3) VALUES (4, 5, 6);
INSERT INTO my_table (col1, col2, col3) VALUES (1, 2, 3);
INSERT INTO my_table (col1, col2, col3) VALUES (4, 5, 6);
INSERT INTO my_table (col1, col2, col3) VALUES (7, 8, 9);
```

1. Database client makes a request to Postgres
2. Postgres evaluates SQL statement
3. Postgres performs operation
4. Postgres prepares result
5. Postgres sends back result
6. Database client processes result
7. Repeat steps 1 - 6 four more times

Reduce round trips: Example

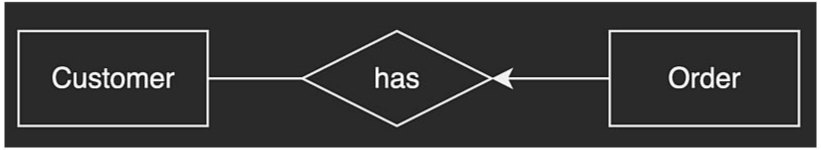
```
INSERT INTO my_table (col1, col2, col3) VALUES (1, 2, 3);
INSERT INTO my_table (col1, col2, col3) VALUES (4, 5, 6);
INSERT INTO my_table (col1, col2, col3) VALUES (1, 2, 3);
INSERT INTO my_table (col1, col2, col3) VALUES (4, 5, 6);
INSERT INTO my_table (col1, col2, col3) VALUES (7, 8, 9);
```

↓

```
INSERT INTO my_table (col1, col2, col3) VALUES
(1, 2, 3), (4, 5, 6), (1, 2, 3), (4, 5, 6), (7, 8, 9);
```

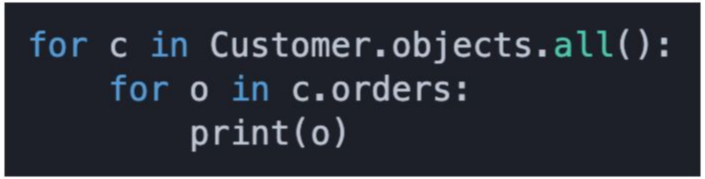
1. Database client makes a request to Postgres
2. Postgres evaluates SQL statement
3. Postgres performs operation
4. Postgres prepares result
5. Postgres sends back result
6. Database client processes result
7. ~~Repeat steps 1 - 6 four more times~~

N + 1 Query



Common pitfall when using ORM

Task: Print all orders for each customer

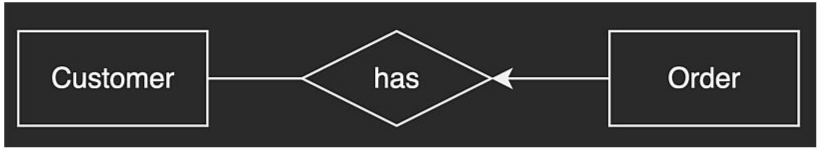


```
for c in Customer.objects.all():
    for o in c.orders:
        print(o)
```

Database receives swell of queries

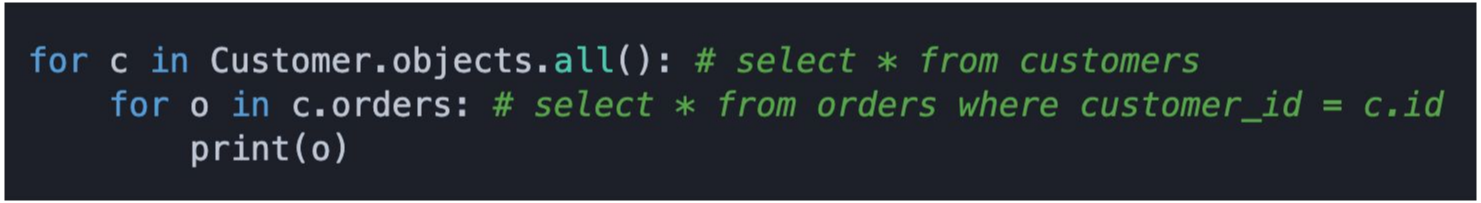
What went wrong?

N + 1 Query



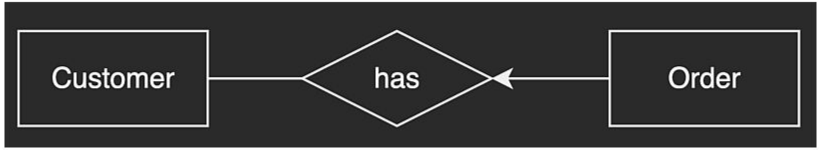
Common pitfall when using ORM

Task: Print all orders for each customer



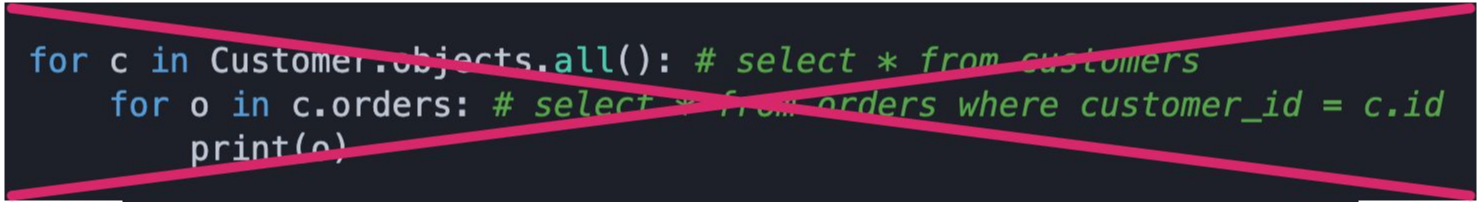
```
for c in Customer.objects.all(): # select * from customers
    for o in c.orders: # select * from orders where customer_id = c.id
        print(o)
```

N + 1 Query

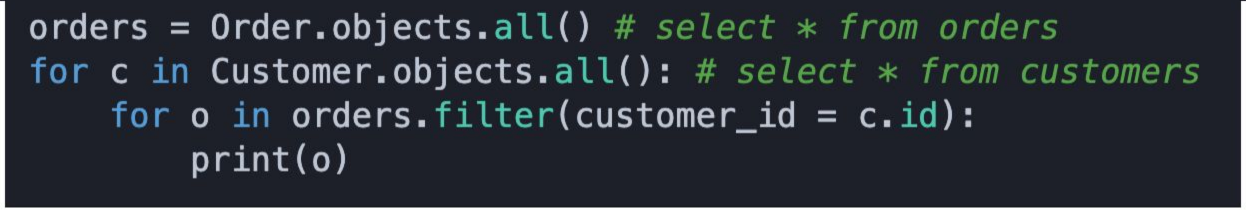


Common pitfall when using ORM

Task: Print all orders for each customer



```
for c in Customer.objects.all(): # select * from customers
    for o in c.orders: # select * from orders where customer_id = c.id
        print(o)
```



```
orders = Order.objects.all() # select * from orders
for c in Customer.objects.all(): # select * from customers
    for o in orders.filter(customer_id = c.id):
        print(o)
```

Final points

Optimize the slowest bottlenecks first

Avoid storing values as HTML, XML, JSON

Establish data retention policy
Delete or archive obsolete data

Database modeling only goes so far
Data types, normalization

Query formation must be thoughtful
Keep queries simple, reduce round-trips, avoid N + 1 pitfall