SQL:
A Trojan horse hiding a decathlon of complexities

ACM SIGMOD DataEd 2023

Toni Taipalus
University of Jyväskylä, Finland
toni.taipalus@jyu.fi

How complexities are hidden

> SQL is a relatively easy language to learn. Very similarly structured to the English language, SQL can be understood quite quickly by many people. It's an elegant solution to searching for data in structured databases.

codecademy.com

> SQL is intuitive, practical, and easy to use. Even with no background in technology, you can master the fundamentals of the language. SQL uses a syntax that is very similar to English, which means that learning SQL is a smooth process.

careerkarma.com

> Because SQL is a relatively simple language, learners can expect to become familiar with the basics within two to three weeks. That said, if you're planning on using SQL skills at work, you'll probably need a higher level of fluency.

bootcamp.berkeley.edu

How complexities are hidden

- Theory behind relational databases has solid mathematical foundations.
- Implementations are mature.

- A domain-specific language used in simple environments.
- Query constructs, SQL syntax, etc. appear simple.

- Effectively a part of every higher education computing curricula.
- Abundance of textbooks, online tutorials, forum Q&A…

- Professionals have learned to work with (and around) the quirks of SQL.
- For a novice, each discrepancy, strange convention, etc. is a complexity.

The underlying principles

1. the role of relational theory

- Exceptionally well-defined:
  - Formal definitions of data structures (the relational model)
  - Formal definitions of operations (set theory operations)
  - Formal definitions of design principles (normalization theory)

1. the role of relational theory

Yes, but

1. the role of relational theory

- Normalization is complex.

transitional dependency
the complexity of business domains
key attribute
primary key
(true) subsets and (true) supersets
full functional dependency
normal forms
Armstrong's axioms
set theory
functional dependency
join dependency
candidate key
multi-valued dependency
superkey
trivial and nontrivial dependency
"what was that boycott normal form again and what are we boycotting?"

- Normalization is applied to various degrees or not applied at all.
- The Standard defines (and RDBMSs implement) non-atomic data types.

2. data demand agnosticism

- Follow normalization theory, and the database can satisfy effectively any demand for data, given that you have that data in your database.

2. data demand agnosticism

Yes, but

2. data demand agnosticism

PostgreSQL (SQL)

```
SELECT *
FROM orders;
```

```
SELECT c.*
FROM customers c
JOIN orders o ON (c.id = o.cust_id);
```

```
SELECT c.*
FROM customers c
JOIN orders o        ON (c.id = o.cust_id)
JOIN order_lines ol ON (o.line_id = ol.id)
JOIN products p     ON (ol.prod_id = p.id)
WHERE EXTRACT(YEAR FROM o.order_date) = 2023
AND p.itemname ILIKE '%toaster%';
```

```
SELECT c.*
FROM customers c
WHERE EXISTS
 (SELECT *
 FROM orders o
```

Cassandra (CQL)

```
SELECT *
FROM orders;
```

```
SELECT *
FROM customers_with_orders;
```

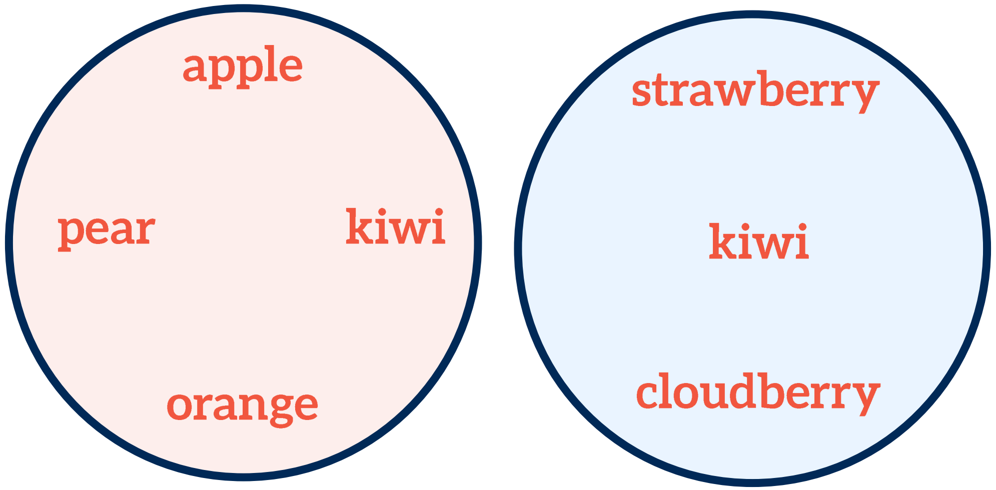
```
SELECT *
FROM this_year_cust_with_toasters;
```

```
SELECT *
FROM this_year_cust_with_<snip>
     100_toasters_but_no_laptops;
```

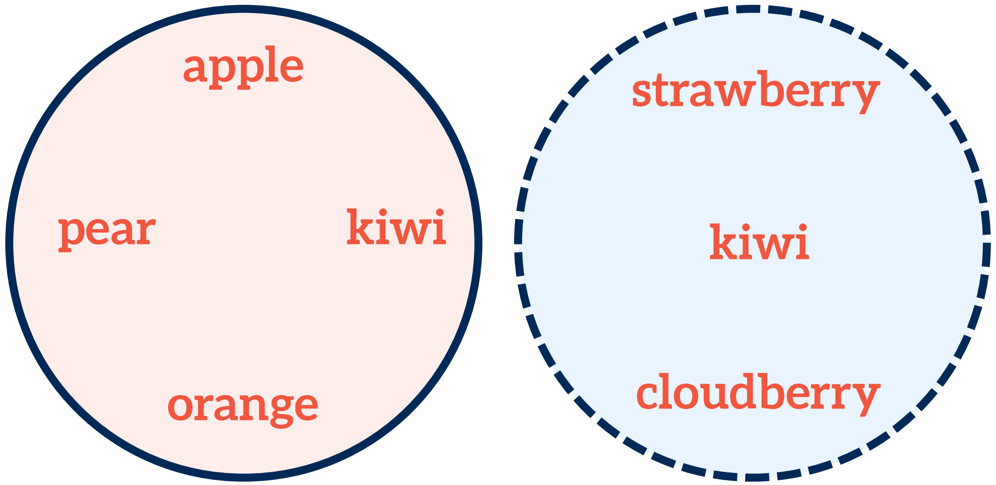
3. sets and operations

- To the degree set theory is used in SQL, the operations are intuitive.

which elements belong to both sets?



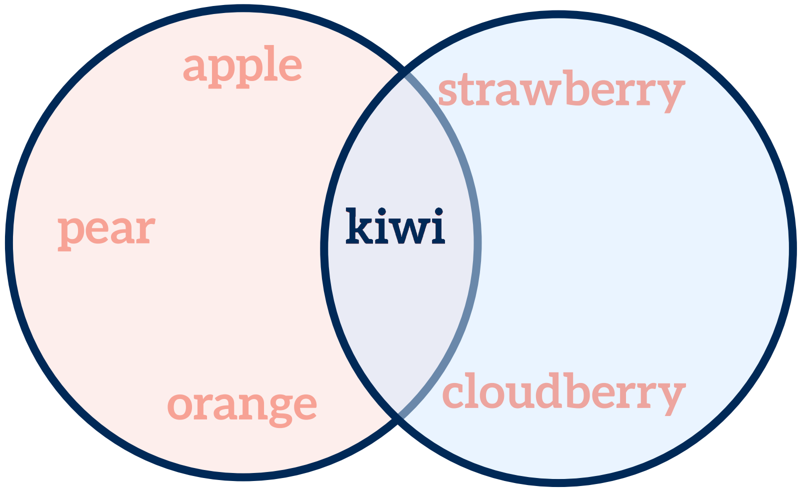
which elements that are part of the left set are not present in the right set?



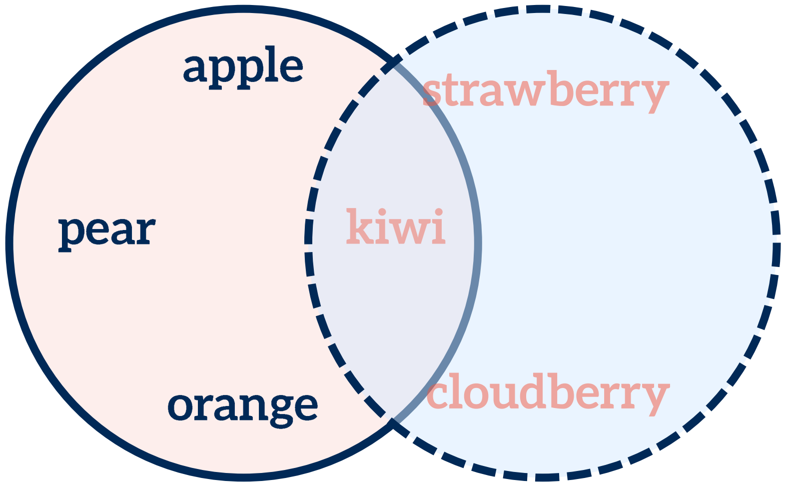
3. sets and operations

- To the degree set theory is used in SQL, the operations are intuitive.

which elements belong to both sets?



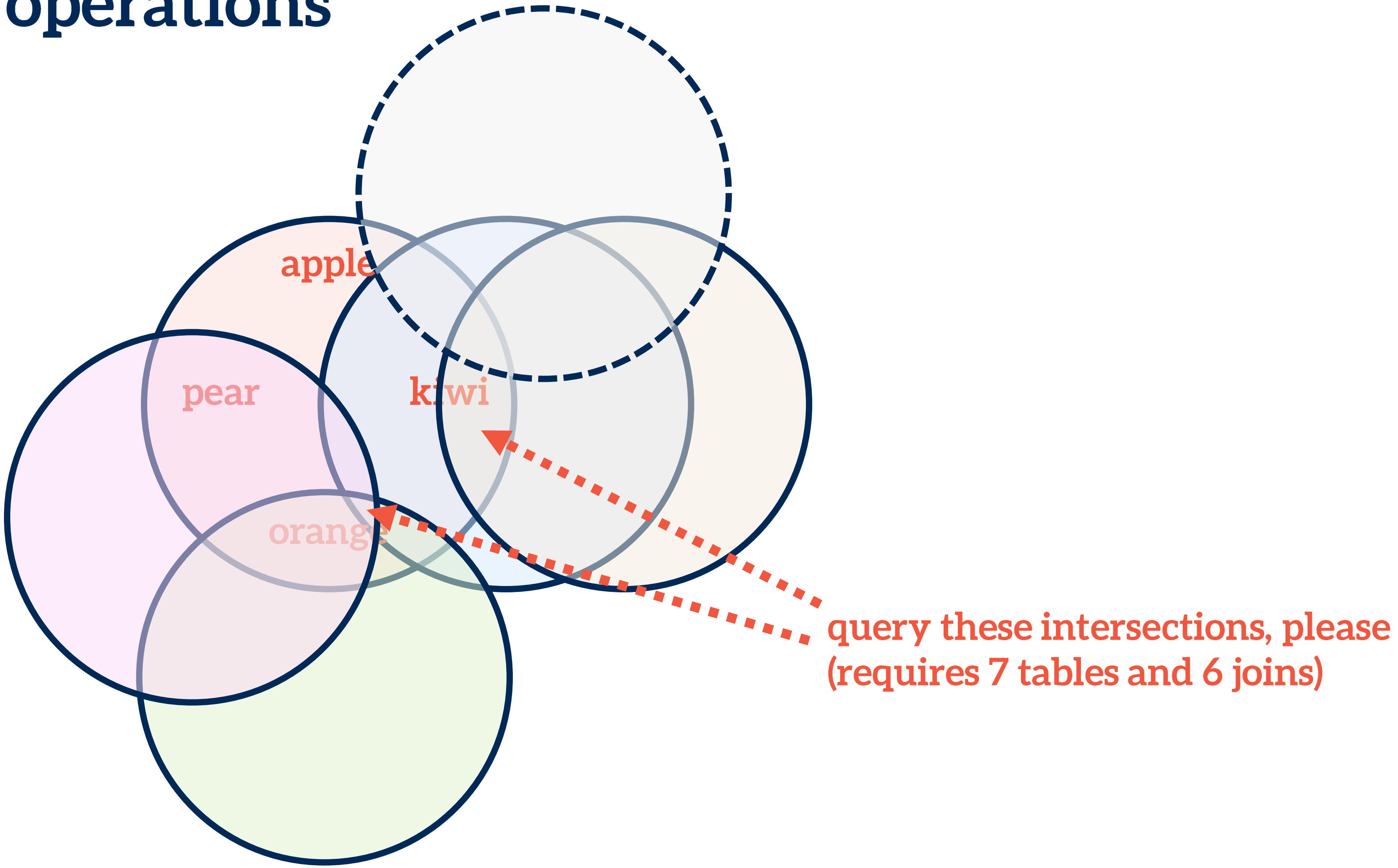
which elements that are part of the left set are not present in the right set?



3. sets and operations

Yes, but

3. sets and operations



The language

4. imperative or declarative

- What versus how.
- SQL's syntax is simple and looks like English.
- Declarative nature sounds user-friendly, accessible and high-level.

```
SELECT surname
  FROM students
 WHERE age > 20;
```

Select the surnames
of students
who are older than 20 years.

4. imperative or declarative

Yes, but

4. imperative or declarative

```
WITH prices AS (
    SELECT EXTRACT(MONTH FROM orderdate) AS month
         , EXTRACT(YEAR FROM orderdate)  AS year
         , SUM(totalprice)               AS price
    FROM orders
    GROUP BY month, year
)
SELECT prices.year
     , prices.month
     , prices.price
     , SUM(prices.price) OVER (
         PARTITION BY year
         ORDER BY month
       ) AS price_cumulative
FROM prices
ORDER BY year ASC, month ASC;
```

How declarative is this?

Select the sums and
the cumulative sums of prices
of ordered products
yearly and monthly.

5. a myriad of choices

- Operators to ease some arduous query constructs
  - IN, BETWEEN, OVERLAPS, etc.
- Multiple alternatives for joining tables
  - JOIN, IN, EXISTS, etc.
- Different approaches to complex query constructs
  - GROUP BY + HAVING instead of NOT EXISTS + NOT EXISTS, etc.

```
SELECT DISTINCT x.A
FROM T1 AS x
WHERE NOT EXISTS
  (SELECT *
  FROM T2 y
  WHERE NOT EXISTS
    (SELECT *
    FROM T1 AS z
    WHERE (z.A=x.A) AND (z.B=y.B)));
```

```
SELECT A
FROM T1
WHERE B IN (SELECT B FROM T2)
GROUP BY A
HAVING COUNT(*) =
  (SELECT COUNT (*) FROM T2);
```

[MG02]

5. a myriad of choices

Yes, but

5. a myriad of choices

- The different ways of writing queries are not always interchangeable.

- Joins with IN and EXISTS behave differently when NULLs are present.
- Where do I put the expressions when I use JOINs?

- When must I use a subquery?
- When can't I use a subquery?

6. strange conventions

- SQL is a high-level language with little syntactical padding.
- Again, SQL statements look a lot like English.

6. strange conventions

Yes, but

6. strange conventions

```
SELECT name
FROM products
WHERE price =
  (SELECT MAX(price)
  FROM products);
```

Why this and not this?

```
SELECT name
FROM products
WHERE price = MAX(price);
```

```
SELECT color, COUNT(*)
FROM products
```

```
GROUP BY color;
```

If this must always be followed by this, why must I write this at all?

```
SELECT a, b, c, d, AVG(e)
FROM products
```

```
GROUP BY a, b, c, d;
```

7. three-valued logic

- (NULL) equals (NULL)
- (NOT NULL) equals (NULL)
- (price = NULL) equals (NULL)

P	Q	P AND Q	P OR Q
True	True	True	True
True	False	False	True
False	False	False	False
True	Unknown	Unknown	True
False	Unknown	False	Unknown

7. three-valued logic

Yes, but

7. three-valued logic

product_id	price
1	10
2	NULL
3	10

- So,
  - SUM(price) must be NULL (it isn't).
  - AVG(price) must be NULL (it isn't).
  - MIN(price) must be NULL (it isn't)…
- Three-valued logic is not suited for relational databases [Ru07, Da08].
  - We need a separate operator (IS)…
  - …and functions (COALESCE, NULLIF) to check for NULLs.
  - GROUP BY groups NULLs to the same group.
  - Aggregate functions disregard NULLs.
  - Joins (JOIN, EXISTS) operate using two-valued logic…

The environments

8. dialects

- The SQL Standard makes the language portable across different systems.

8. dialects

Yes, but

8. dialects

```
SELECT *
FROM reservations
WHERE (start_time, end_time) OVERLAPS (:start, :end);
```

Does not work in MySQL

```
...FOREIGN KEY (cust_id) REFERENCES customers (id)
  ON UPDATE CASCADE
  ON DELETE CASCADE;
```

Does not work in Oracle Database

```
FULL OUTER JOIN customers ON (cust_id ...
```

Does not work in SQLite

```
...WHERE EXTRACT(YEAR FROM start_time) = 2023;
```

Does not work in SQL Server

```
SELECT nationality, COUNT(*)
FROM customers
GROUP BY id;
```

Does not work in PostgreSQL

9. error messages

- RDBMSs that implement SQL are mature, and
- developed by diverse teams of experts with hefty budgets.

- Human-computer interaction has come a long way since the 1970s.

9. error messages

Yes, but

9. error messages

```
SELECT *
FROM customers c
WHERE EXISTS
  (SELECT *
  FROM orders o
  WHERE c.id = o.cust_id
  ORDER BY o.cust_id);
```

```
ORA-00907: missing right parenthesis
```

Oracle 7i (1992)

```
ORA-00907: missing right parenthesis
```

Oracle 23c (2023)

[Ta23]

An error occurred in your query

```
ERROR: division by zero
```

```
-- customers by product group:
WITH pg AS (
    SELECT p.groupname                    AS groupname
         , COUNT(DISTINCT o.customerid) AS num_cust
    FROM orders o
    RIGHT JOIN orderlines ol
        ON (o.orderid = ol.orderid)
    LEFT JOIN products p
        ON (ol.productid = p.productid)
    GROUP BY 1
)
-- customers by state:
, sta AS (
    SELECT o.state                        AS state
         , COUNT(DISTINCT o.customerid) AS num_cust
    FROM orders o
    GROUP BY 1
)
-- customers by product group and state:
, pg_sta AS (
    SELECT p.groupname                    AS groupname
         , o.state                        AS state
         , COUNT(DISTINCT o.customerid) AS num_cust
    FROM orders o
    RIGHT JOIN orderlines ol
        ON (o.orderid = ol.orderid)
    LEFT JOIN products p
        ON (ol.productid = p.productid)
    GROUP BY 1, 2
)
-- expected values:
, exp AS (
    SELECT pg_sta.state
         , pg_sta.groupname
         , pg_sta.num_cust
         , pg.num_cust * sta.num_cust /
           (SELECT COUNT(DISTINCT customerid)
            FROM orders) AS expected
    FROM pg_sta
    LEFT OUTER JOIN sta
        ON (pg_sta.state = sta.state)
    LEFT OUTER JOIN pg
        ON (pg_sta.groupname = pg.groupname)
)
-- chi square:
SELECT state
     , groupname
     , num_cust
     , expected
     -- chi square calculation:
     , POWER(num_cust - expected, 2) / expected AS chisquare
FROM exp
ORDER BY chisquare DESC
LIMIT 10;
```

Yes, but

10. lack of error messages

- RDBMSs have sophisticated compilers and query optimizers.

10. lack of error messages

Yes, but

10. lack of error messages

```
SELECT fname, sname
FROM customers
WHERE age > 20 AND age < 20;
```

```
fname | sname
------+-------
(0 rows)
```

```
<EXPLAIN ANALYZE>
SELECT fname, sname
FROM customers
WHERE age > 20 AND age < 20;
```

```
never executed
```

PostgreSQL

```
filter (NULL IS NOT NULL)
```

Oracle Database

```
Impossible WHERE
```

MySQL

Pedagogical parting thoughts

- Relational model: first informally, then formally.
- Visualize queries [DG11, MF21, Ta19].
- Do not treat SQL like a natural language.

- Teach one SQL dialect.
- Use a DBMS that
  - tries to conform to SQL Standard and
  - has (relatively) effective error messages [TG21].

- Use an engaging exercise database [TM23].

References and thank you

- References

  - [DG11] Danaparamita & Gatterbauer (2011). QueryViz: Helping Users Understand SQL Queries and Their Patterns. EDBT'11.
  - [Da08] Date (2008). A Critique of Claude Rubinson's Paper Nulls, Three-Valued Logic, and Ambiguity in SQL: Critiquing Date's Critique. SIGMOD Rec.
  - [MG02] Matos & Grasser (2002). A Simpler (and Better) SQL Approach to Relational Division. JISE.
  - [MF21] Miedema & Fletcher (2021). SQLVis: Visual Query Representations for Supporting SQL Learners. VL/HCC'21.
  - [Ru07] Rubinson (2007). Nulls, Three-Valued Logic, and Ambiguity in SQL: Critiquing Date's Critique. SIGMOD Rec.
  - [Ta19] Taipalus (2019). A notation for planning SQL queries. JISE.
  - [Ta23] Taipalus (2023). Query execution plans and semantic errors: Usability and educational opportunities. ACM CHI'23.
  - [TG21] Taipalus, Grahn & Ghanbari (2021). Error messages in relational database management systems: a comparison of effectiveness, usefulness and user confidence. JSS.
  - [TM23] Taipalus, Miedema & Aivaloglou (2023). Engaging databases for data systems education. ITiCSE'23.
  - "Yes, but" images from DALL·E, "Gustave Doré portrait style image of a confused [person/cat/dachshund/corgi/etc]."
  - "Trojan horse" images from DALL·E, "A realistic painting of a trojan horse, with small silhouettes of people pointing at it in awe."

- Thank you