SQL 101

Ethan Kent

Spoonflower

January 5, 2021

What is SQL?

- SQL is a programming language for interacting with relational databases.
- It stands for *S*tructured *Q*uery *L*anguage.[1]
- It was invented in 1974, and became a standard of ANSI and ISO in the late-1980s.
- Standards for SQL are now defined by ISO/IEC, though it appears that no actual SQL versions are compliant with all and only the ISO/IEC standards.

[1] Or maybe it doesn't: "It's a common misconception that *SQL* stands for *structured query language*; it stands for S-Q-L and nothing else. Why? Because ISO says so." Chris Fehily, *SQL Database Programming* at loc'n 222 (2015 Kindle ed.)

What is a Relational Database?

- A database (imperfectly) implements a *relational model* of data, which is a mathematical model describing certain data structures and relationships.
- From an practical engineering perspective, a relational database has rows and columns, where each row is an entry in the database; and each column is a bit of data about that entry that matches the type required for that column.
- A relational database contrasts with NoSQL databases like MongoDB, in which users can insert free-form data that does not comprise rows of data in typed columns.

What's up with MySQL, PostgreSQL, Microsoft SQL Server, Oracle, IBM DB2, . . .

- The situation is similar to the existence of many programming languages aimed at the same problem domains.
- Some are open-source, some aren't.
- Some are more compliant with ISO/IEC standards (e.g. PostgreSQL); some are less so (e.g. SQLite).
- Some are part of ecosystems, are cloud-based/SaaS, etc. (e.g. Microsoft SQL Server, Amazon RDS, Google Cloud SQL).

What does “relational” mean?

Spreadsheet analogy

You’ve probably encountered this problem if you’ve used a spreadsheet to keep track of things. It starts out simple:

Person	Age
Ethan	39
Madigan	39
Belle	9
Milo	7
Clara	3

What does “relational” mean?

Spreadsheet analogy (cont’d)

Now we add a bit:

Person	Age	Favorite Dessert
Ethan	39	Oreo Cookies
Madigan	39	Klondike Bar
Belle	9	Mint Chip Ice Cream
Milo	7	Cookie Dough Ice Cream
Clara	3	Chocolate Ice Cream

What does “relational” mean?

Spreadsheet analogy (cont’d)

And now it turns out one person can’t decide and has *two* favorites. We could do this:

Person	Age	Favorite Dessert
Ethan	39	Oreo Cookies, Mint Chip Ice Cream
Madigan	39	Klondike Bar
Belle	9	Mint Chip Ice Cream
Milo	7	Cookie Dough Ice Cream
Clara	3	Chocolate Ice Cream

Or maybe we try this:

Person	Age	Favorite Dessert #1	Favorite Dessert # 2
Ethan	39	Oreo Cookies	Mint Chip Ice Cream
Madigan	39	Klondike Bar	
Belle	9	Mint Chip Ice Cream	
Milo	7	Cookie Dough Ice Cream	
Clara	3	Chocolate Ice Cream	

What does “relational” mean?

Spreadsheet analogy (cont’d)

What’s wrong with this?

- The comma-separated list is bad because—
  - We can’t sort or group things together, and
  - The value has to be manipulated to be used.
- The use of additional columns is bad because—
  - We have to keep adding columns based on the data, so we’re never sure we’re done with the structure of the database; and
  - We are treating like things differently: columns mean the same thing, but are distinct.

What does "relational" mean?

Spreadsheet analogy (cont'd)

So we do this:

Person	Age	Favorite Dessert
Ethan	39	Oreo Cookies
Ethan	39	Mint Chip Ice Cream
Madigan	39	Klondike Bar
Belle	9	Mint Chip Ice Cream
Milo	7	Cookie Dough Ice Cream
Clara	3	Chocolate Ice Cream

What does “relational” mean?

Spreadsheet analogy (cont’d)

And now we want to have more detail about the desserts:

Person	Age	Favorite Dessert	Dessert Type
Ethan	39	Oreos	Cookie
Ethan	39	Mint Chip	Ice Cream
Madigan	39	Klondike Bar	Ice Cream Novelty
Belle	9	Mint Chip	Ice Cream
Milo	7	Cookie Dough	Ice Cream
Clara	3	Chocolate	Ice Cream

What does “relational” mean?

Spreadsheet analogy (cont’d)

At this point we might feel queasy:

- We’re repeating ourselves—
  - Once we know the person is *Ethan*, we know we have a 39-year-old person.
  - Once we know the dessert is *Mint Chip*, we know we have an *Ice Cream*.
- We have have data about different stuff in the same place.

What does “relational” mean?

Spreadsheet analogy (cont’d)

Repeating ourselves is more than an inconvenience. Consider this:

Person	Age	Favorite Dessert	Dessert Type
Ethan	39	Oreos	Cookie
Ethan	38	Mint Chip	Ice Cream
Madigan	39	Klondike Bar	Ice Cream Novelty
Belle	9	Mint Chip	Candy Bar
Milo	7	Cookie Dough	Ice Cream
Clara	3	Chocolate	Ice Cream

- Are these two desserts with the same name (“Mint Chip”)?
- Are there two *Ethan*s?
- Are there mistakes?
- If there are mistakes, is Ethan 38 or 39? Is Mint Chip an Ice Cream or a Candy Bar?

What does "relational" mean?

Spreadsheet analogy (cont'd)

So let's split things into different tables. And let's use some arbitrary, unique "ID"s, so we can talk about a particular row without having to use some of the data (that could change):

ID	Person	Age
1	Ethan	39
2	Madigan	39
3	Belle	9
4	Milo	7
5	Clara	3

ID	Dessert	Dessert Type
1	Oreos	Cookies
2	Mint Chip	Ice Cream
3	Klondike Bar	Ice Cream Novelty
4	Cookie Dough	Ice cream
5	Chocolate	Ice Cream

But now how can we show the relationship between these? Wait, did I just say *relationship*? As in *relational database*? Holy cow.

What does “relational” mean?

Spreadsheet analogy (cont’d)

ID	Person	Age
1	Ethan	39
2	Madigan	39
3	Belle	9
4	Milo	7
5	Clara	3

ID	Dessert	Dessert Type
1	Oreos	Cookies
2	Mint Chip	Ice Cream
3	Klondike Bar	Ice Cream Novelty
4	Cookie Dough	Ice cream
5	Chocolate	Ice Cream

Person ID	Dessert ID
1	1
1	2
2	3
3	2
4	4
5	5

What does “relational” mean?

Spreadsheet analogy (cont’d)

Did you notice the error that will bite us later? Cookie Dough is an “Ice cream,” not an “Ice Cream.” Let’s fix that:

ID	Person	Age
1	Ethan	39
2	Madigan	39
3	Belle	9
4	Milo	7
5	Clara	3

ID	Desert Type
1	Cookie
2	Ice Cream
3	Ice Cream Novelty

ID	Dessert	Dessert Type ID
1	Oreos	1
2	Mint Chip	2
3	Klondike Bar	3
4	Cookie Dough	2
5	Chocolate	2

Person ID	Dessert ID
1	1
1	2
2	3
3	2
4	4
5	5

What does “relational” mean?

Spreadsheet analogy (cont’d)

What does any of this have to do with anything? Well, you’ve just seen examples of:

- Relational database design
- Foreign keys
- Primary keys
- Composite primary keys
- Join tables
- First and Third Normal Form

Primary keys

ID	Person	Age
1	Ethan	39
2	Madigan	39
3	Belle	9
4	Milo	7
5	Clara	3

ID	Desert Type
1	Cookie
2	Ice Cream
3	Ice Cream Novelty

ID	Dessert	Dessert Type ID
1	Oreos	1
2	Mint Chip	2
3	Klondike Bar	3
4	Cookie Dough	2
5	Chocolate	2

Person ID	Dessert ID
1	1
1	2
2	3
3	2
4	4
5	5

Foreign keys

ID	Dessert	Dessert Type ID
1	Oreos	1
2	Mint Chip	2
3	Klondike Bar	3
4	Cookie Dough	2
5	Chocolate	2

Person ID	Dessert ID
1	1
1	2
2	3
3	2
4	4
5	5

Composite primary keys

Person ID	Dessert ID
1	1
1	2
2	3
3	2
4	4
5	5

Join tables

Person ID	Dessert ID
1	1
1	2
2	3
3	2
4	4
5	5

First Normal Form

Don't do either of these things (columns must be "atomic" and there mustn't be "repeating groups"):

Person	Favorite Dessert
Ethan	Oreos, Mint Chip
Madigan	Klondike Bar
Belle	Mint Chip
Milo	Cookie Dough

Person	Favorite Dessert #1	Favorite Dessert # 2
Ethan	Oreos	Mint Chip
Madigan	Klondike Bar	
Belle	Mint Chip	
Milo	Cookie Dough	

Third Normal Form

Don’t do this:

- If we know *Ethan*, then we automatically know *39*; and
- If we know *Mint Chip*, then we automatically know *Ice Cream*:[2]

Person	Age	Favorite Dessert	Dessert Type
Ethan	39	Oreos	Cookie
Ethan	39	Mint Chip	Ice Cream
Madigan	39	Klondike Bar	Ice Cream Novelty
Belle	9	Mint Chip	Ice Cream
Milo	7	Cookie Dough	Ice Cream
Clara	3	Chocolate	Ice Cream

[2] In First and Second Normal Forms and no “transitive dependencies.” Second Normal Form has to do with composite primary keys, but it’s impossible with the tables we’ve looked at so far.

Create some tables

```
CREATE TABLE people (
  id SERIAL NOT NULL,
  name TEXT NOT NULL UNIQUE,
  age INTEGER NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE dessert_types (
  id SERIAL NOT NULL,
  name TEXT NOT NULL UNIQUE,
  PRIMARY KEY (id)
);
```

Create some more tables

```
CREATE TABLE desserts (
  id SERIAL NOT NULL,
  name TEXT NOT NULL UNIQUE,
  dessert_type_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (dessert_type_id)
    REFERENCES dessert_types(id)
);

CREATE TABLE people_desserts (
  person_id INTEGER NOT NULL,
  dessert_id INTEGER NOT NULL,
  PRIMARY KEY (person_id, dessert_id),
  FOREIGN KEY (person_id) REFERENCES people(id),
  FOREIGN KEY (dessert_id) REFERENCES desserts(id)
);
```

Insert some data

```
INSERT INTO people (name, age) VALUES
  ('Ethan', 39),
  ('Madigan', 39),
  ('Belle', 9),
  ('Milo', 7),
  ('Clara', 3);

INSERT INTO dessert_types (name) VALUES
  ('Cookie'),
  ('Ice Cream'),
  ('Ice Cream Novelty');
```

Now what?

```
INSERT INTO desserts (dessert_name, dessert_type_id) VALUES
  — ('Oreos', ???),
```

Queries!

```
INSERT INTO desserts (name, dessert_type_id) VALUES
  ('Oreos',
    (SELECT id FROM dessert_types
     WHERE name = 'Cookie')),

  ('Mint Chip',
    (SELECT id FROM dessert_types
     WHERE name = 'Ice Cream')),

  ('Klondike Bar',
    (SELECT id FROM dessert_types
     WHERE name = 'Ice Cream Novelty')),

  ('Cookie Dough',
    (SELECT id FROM dessert_types
     WHERE name = 'Ice Cream')),

  ('Chocolate',
    (SELECT id FROM dessert_types
     WHERE name = 'Ice Cream'));
```

Not very DRY. How about a CTE?

```
WITH
  cookie_id AS
    (SELECT id FROM dessert_types
     WHERE dessert_type = 'Cookie'),
  ice_cream_id AS
    (SELECT id FROM dessert_types
     WHERE dessert_type = 'Ice Cream'),
  ice_cream_novelty_id AS
    (SELECT id FROM dessert_types
     WHERE dessert_type = 'Ice Cream Novelty')
INSERT INTO desserts (dessert_name, dessert_type_id) VALUES
  ('Oreos', (SELECT id FROM cookie_id)),
  ('Mint Chip', (SELECT id FROM ice_cream_id)),
  ('Klondike Bar', (SELECT id FROM ice_cream_novelty_id)),
  ('Cookie Dough', (SELECT id FROM ice_cream_id)),
  ('Chocolate', (SELECT id FROM ice_cream_id));
```

Not very DRY. How about a CTE?

Not with MySQL though.

Okay but we're getting ahead of ourselves!

We're doing CTEs and we haven't even done queries. What is wrong with you?

Okay but we're getting ahead of ourselves!

There's a bootstrapping issue: we need a database to do some database stuff. It's like languages where doing "Hello world" is hard.[3]

[3] I'm looking at you Haskell.

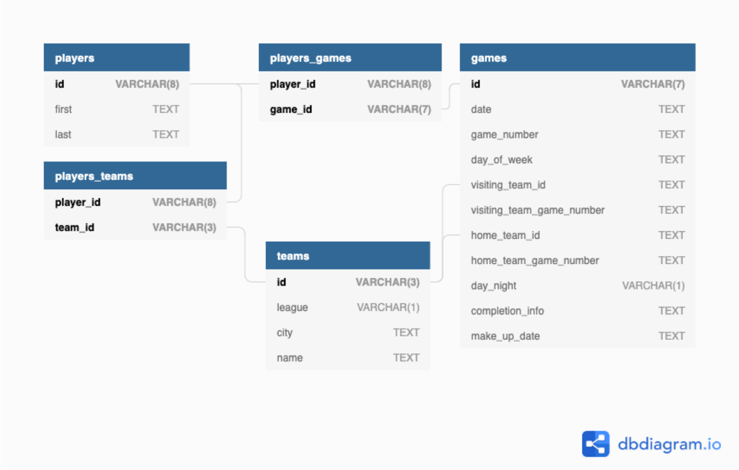
Baseball example

For our German friends, baseball is an American sport that's a bit like cricket, though actually comprehensible. There are 30 teams divided into two leagues,[4] and each team plays 162 games per year. Because of the number of games and long history, baseball is big among statisticians.

[4] Technically, only one of the leagues plays baseball; the other plays a game that appears on the surface like baseball but that is not actually baseball, as it involves the Designated Hitter rule.

Baseball example

I have a rudimentary database of 2016 MLB info: just games, teams, and players.



players	
id	VARCHAR(8)
first	TEXT
last	TEXT

players_teams	
player_id	VARCHAR(8)
team_id	VARCHAR(3)

players_games	
player_id	VARCHAR(8)
game_id	VARCHAR(7)

teams	
id	VARCHAR(3)
league	VARCHAR(1)
city	TEXT
name	TEXT

games	
id	VARCHAR(7)
date	TEXT
game_number	TEXT
day_of_week	TEXT
visiting_team_id	TEXT
visiting_team_game_number	TEXT
home_team_id	TEXT
home_team_game_number	TEXT
day_night	VARCHAR(1)
completion_info	TEXT
make_up_date	TEXT

Baseball example

Let's figure out the following:

1. What player has the id 'dillp101'?
2. How many teams were there in 2016?
3. How many players played in 2016?
4. How many games did Anthony Rizzo play in 2016?
5. What is the commonest first name in MLB history?
6. List all players in the database and include all their 2016 teams.
7. What player played for the most teams (choose only one if it's a tie), and what teams were they?
8. From those players who played at least 10 total games in 2016, what player played the highest percentage of night games?

Baseball example, answer 1

```
SELECT * FROM players WHERE id = 'dillp101';
```

Baseball example, answer 2

```
SELECT COUNT(*) FROM teams;
```

Baseball example, answer 3

```
SELECT COUNT(DISTINCT player_id) FROM players_teams;
```

Baseball example, answer 4

```sql
SELECT COUNT(pg.game_id)
FROM players player
INNER JOIN players_games pg
        ON player.id = pg.player_id
WHERE first = 'Anthony'
  AND last = 'Rizzo';
```

Baseball example, answer 5

```
SELECT COUNT(first), first
FROM players
GROUP BY first
ORDER BY 1 DESC
LIMIT 1;
```

Baseball example, answer 5

```
SELECT p.first, p.last, t.name
FROM players_teams pt
INNER JOIN teams t
          ON pt.team_id = t.id
RIGHT OUTER JOIN players p
                 ON pt.player_id = p.id;
```

Baseball example, answer 7

```
SELECT sub.first AS "First Name",
       sub.last AS "Last Name",
       team.name AS "Team Name"
FROM (
  SELECT player.id,
         player.first,
         player.last,
         COUNT(pt.team_id) AS tally
  FROM players player
  INNER JOIN players_teams pt
          ON pt.player_id = player.id
  GROUP BY player.id
  ORDER BY tally DESC
  LIMIT 1
) sub
INNER JOIN players_teams pt
        ON pt.player_id = sub.id
INNER JOIN teams team
        ON team.id = pt.team_id;
```

Baseball example, answer 8

```
SELECT player.first,
       player.last,
       CAST(night_games.tally AS FLOAT) / total_games.tally AS "Percent"
FROM ( SELECT pg.player_id, COUNT(pg.player_id) AS tally
       FROM players_games pg
       INNER JOIN games game ON game.id = pg.game_id
       WHERE game.day_night = 'N'
       GROUP BY pg.player_id
) AS night_games INNER JOIN (
       SELECT pg.player_id, COUNT(pg.player_id) AS tally
       FROM players_games pg
       INNER JOIN games game ON game.id = pg.game_id
       GROUP BY pg.player_id
) AS total_games ON night_games.player_id = total_games.player_id
INNER JOIN players player ON player.id = total_games.player_id
WHERE total_games.tally >= 10
ORDER BY "Percent" DESC
LIMIT 1;
```