Databases

Jason Staten

Data

Likes

Stock quotes

Heartbeats

Temperature

Appointments

Bookmarks

Showtimes

Earthquakes

Tweets

Recipes

DNA

GPS coordinates

Data

Store


Quotes
Quotes
ORCHESTRA


WD
SATA | 64MB Cache
WD10EZEX
1.0TB
WD Blue™
Desktop Hard Drive
Blue


VISA

56,000 transactions per second

Retrieve

How many new messages do I have?

Where is the nearest car driver?

Which are the most popular movies?

Database

A database is an organized way to store and retrieve data.

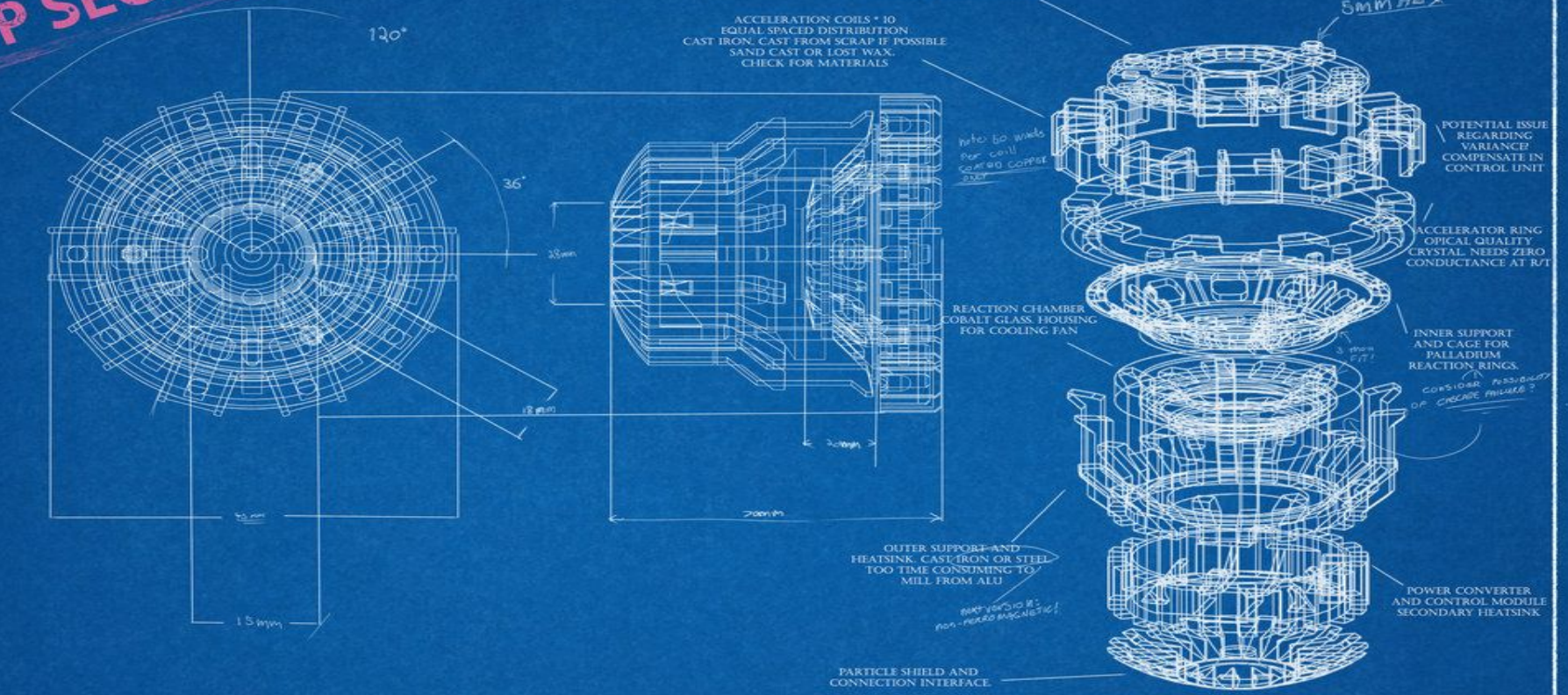
Relational Databases

Tables

id	name	age	country
1	Ted	26	AU
2	Sue	32	US
3	Gwen	24	CA

Schema

TOP SECRET



STARK INDUSTRIES | SAVANTGUARDE | VERSION: 5.34 (FINAL) DESIGNER: | DATE: 4/12/14

```
id INTEGER PRIMARY KEY,
name TEXT,
age INTEGER,
country TEXT
```

INTEGER

REAL

DECIMAL(scale, precision)

TEXT

BLOB

SQL

Structured Query Language

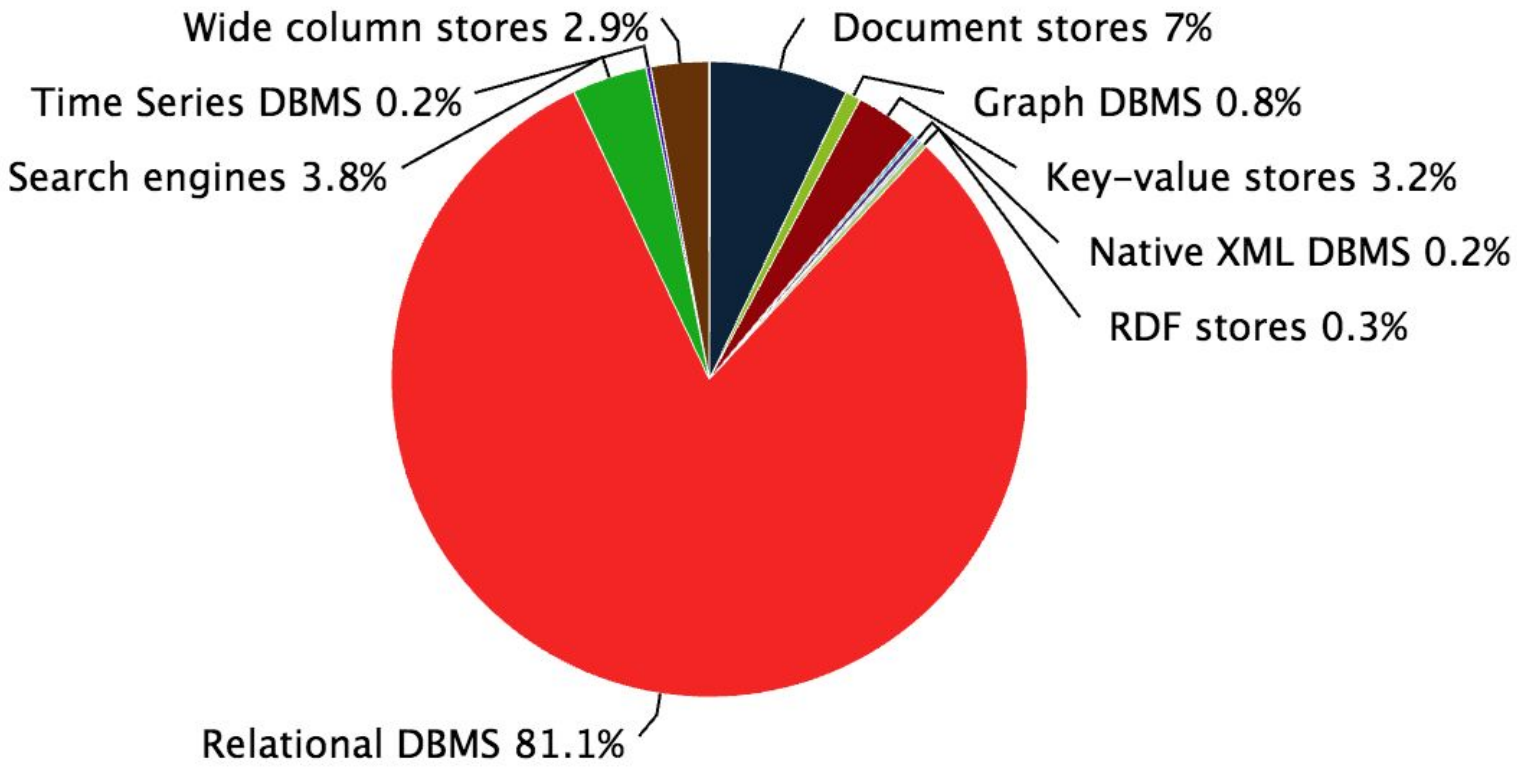
Create tables and rows

Read data out of tables

Update existing data

Delete data from tables

1979



One Language. Any* relational database.

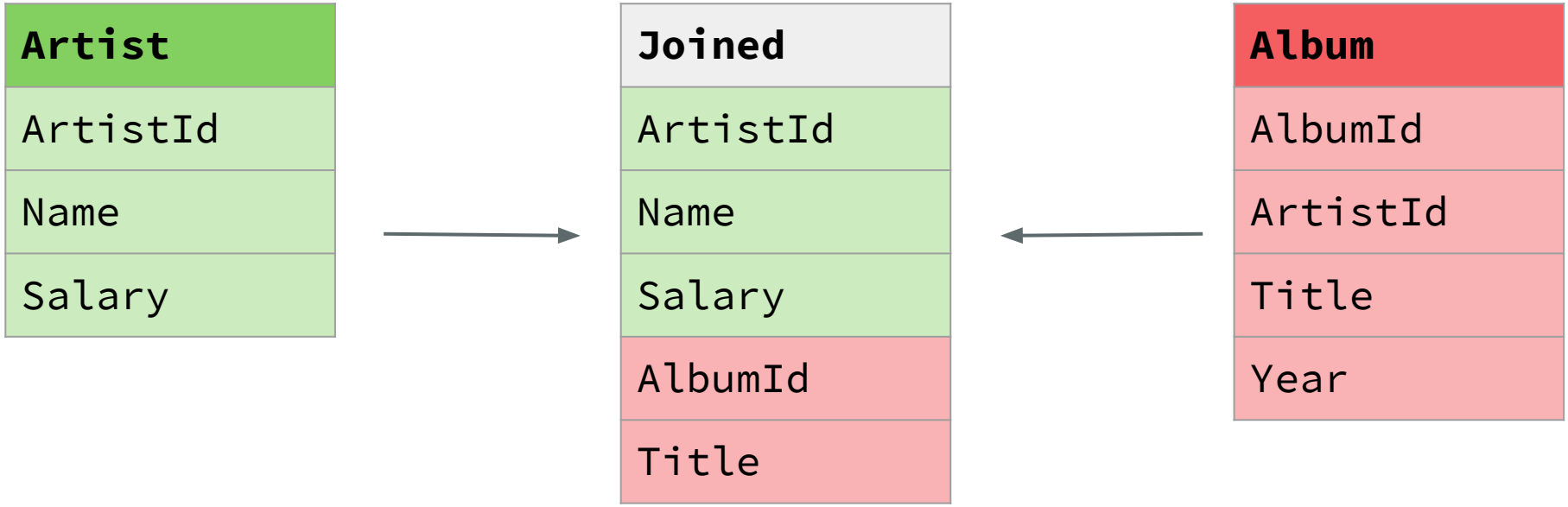
*dialects vary

bit.ly/
devmtnsql

Joins

Artist
ArtistId
Name
Salary

Album
AlbumId
ArtistId
Title
Year




```
SELECT * FROM Album
JOIN Artist
ON Album.ArtistId = Artist.ArtistId
```

GROUP BY

How many customers does each country have?

HAVING

Which country has the most customers?

Which countries have at least 5 customers?

Which albums have earned more than $20?

```
SELECT a.Title FROM Album a
WHERE a.AlbumId IN (
  SELECT t.AlbumId
  FROM InvoiceLine i
  JOIN Track t ON t.TrackId = i.TrackId
  GROUP BY t.AlbumId
  HAVING sum(i.UnitPrice * i.Quantity) > 20
)
```

~~Create tables and rows~~

~~Read data out of tables~~

Update existing data

Delete data from tables

Updates

Updating a table

```
ALTER TABLE injuries
ADD COLUMN painLevel INTEGER
```

```
ALTER TABLE injuries
DROP COLUMN painLevel
```

* DROP COLUMN not implemented in SQLite

Updating rows

```
UPDATE injuries
SET tth = 12
WHERE id = 4
```

Delete

Deleting a table

```
DROP TABLE injuries
```

Deleting rows

```
DELETE FROM injuries
WHERE id = 4
```

Constraints

NOT NULL

```
CREATE TABLE contacts (
   id INTEGER PRIMARY KEY,
   name TEXT NOT NULL,
   email TEXT,
   phone TEXT
)
```

```
NOT NULL constraint failed:
contacts.name
```

Why bother?

Contract

```
var c = {
   id: …,
   name: …,
   phone: …,
   email: …,
}
```

```
c.name.toUpperCase()
```

'ALICE'

```
Uncaught TypeError: Cannot read property 'toUpperCase' of null
```

UNIQUE

```
CREATE TABLE contacts (
   id INTEGER PRIMARY KEY,
   name TEXT NOT NULL,
   email TEXT UNIQUE,
   phone TEXT
)
```

```
INSERT INTO
contacts(name, email)
VALUES('Ron', 'ron@mail.com')

INSERT INTO
contacts(name, email)
VALUES('Bob', 'ron@mail.com')
```

UNIQUE constraint failed: contacts.email

What could be unique?

CHECK

```
CREATE TABLE contacts (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT UNIQUE,
    phone TEXT,
    CHECK (LENGTH(phone) = 10)
)
```

```
INSERT INTO
contacts(name, email, phone)
VALUES(
'Ron',
'ron@mail.com',
'123'
)
```

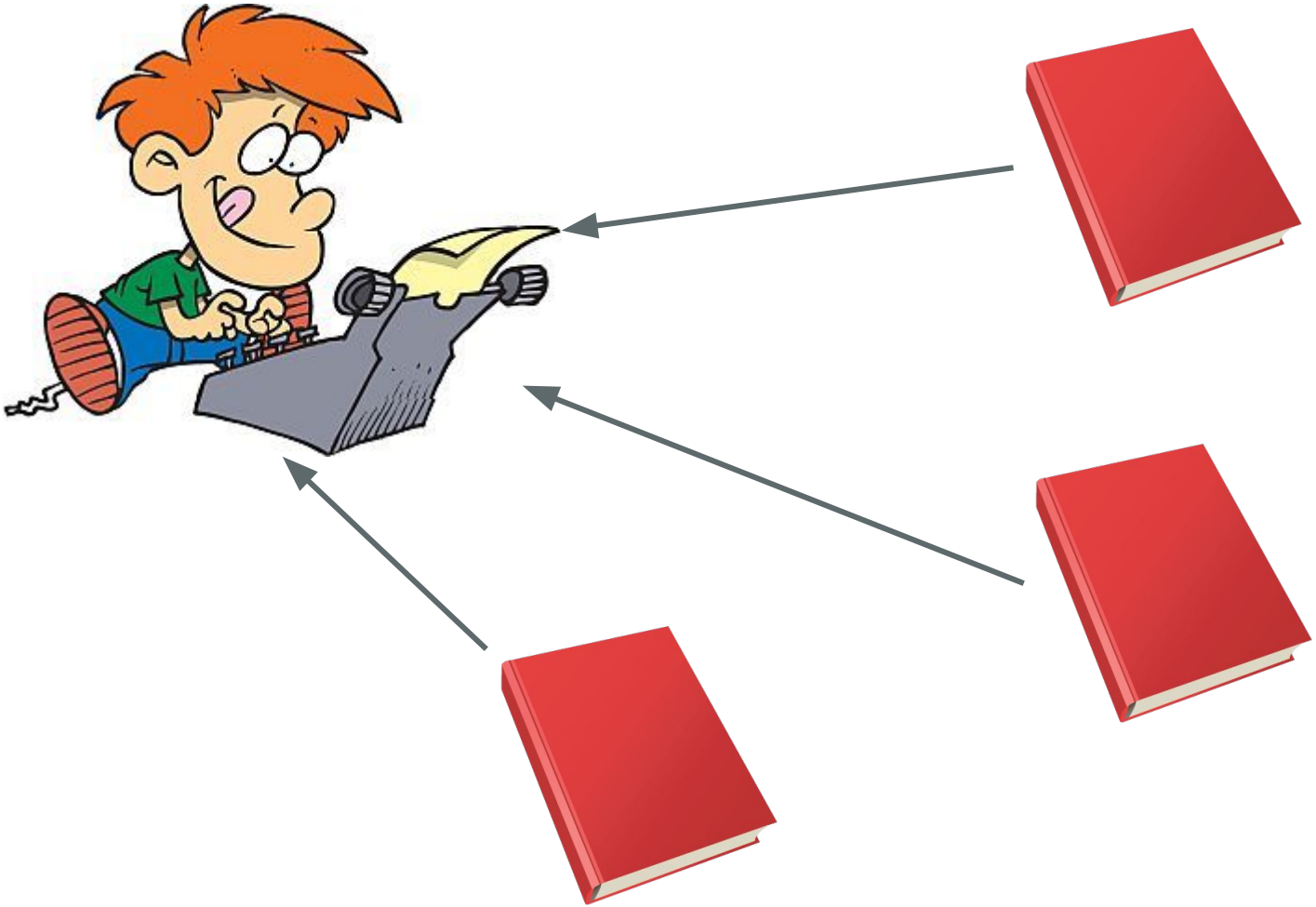
```
CHECK constraint failed:
contacts
```

```
CREATE TABLE triangles (
  id INTEGER PRIMARY KEY,
  a INTEGER NOT NULL,
  b INTEGER NOT NULL,
  c INTEGER NOT NULL,
  CHECK (a + b + c = 180)
)
```

When to use CHECK constraints?


Relationships

many-to-one



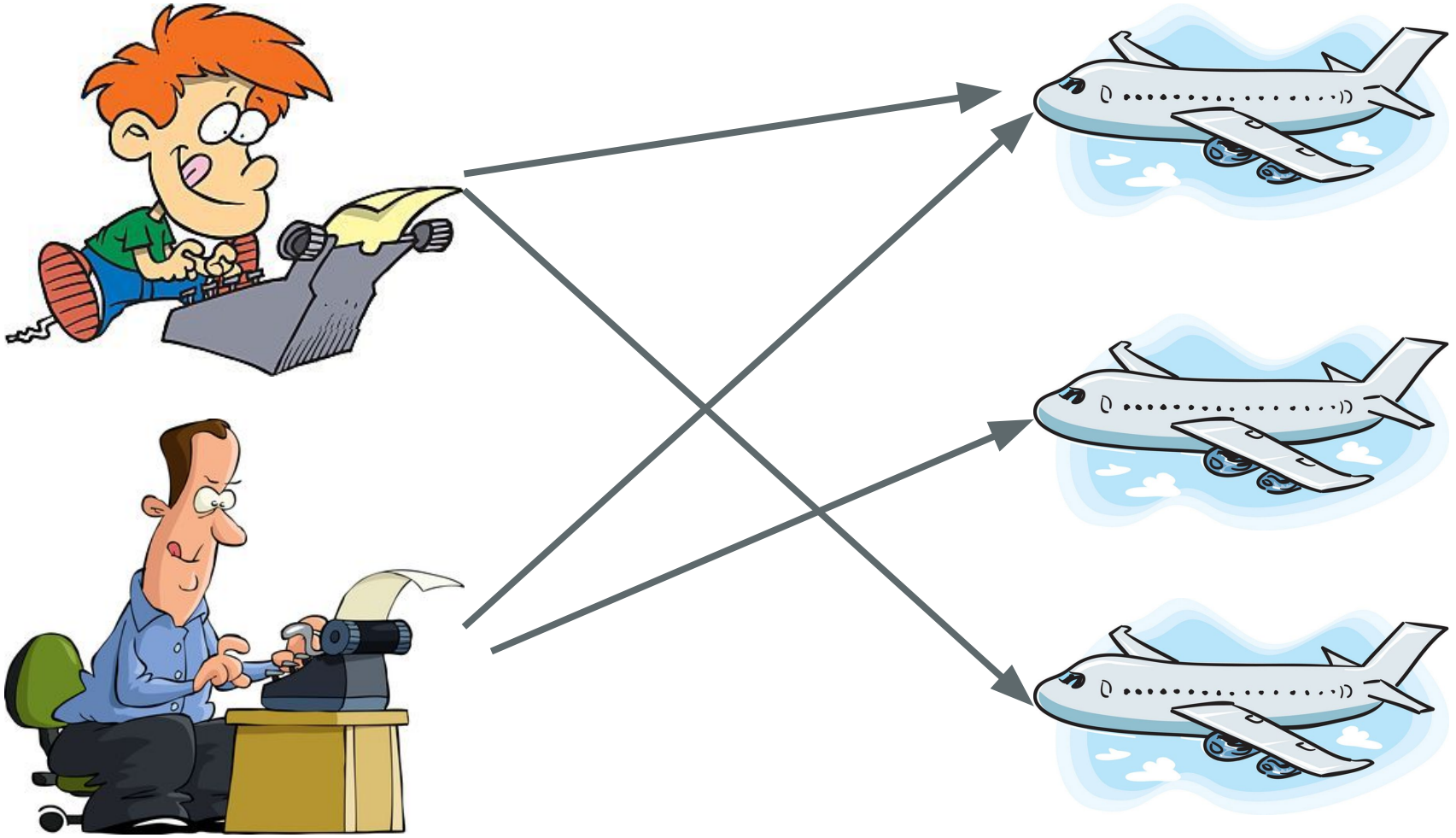
authors

id	name
1	Esteban
2	Mike

books

id	title	authorId
1	A Dark Night	1
2	Warrior King	1
3	How to Ski	2

many-to-many



authors

id	name
1	Esteban
2	Mike

flights

id	airline
8	Delta
10	United

tickets

id	flightId	authorId
1	10	1
2	10	2
3	8	2

one-to-one?

Foreign Keys

many-to-one

```
CREATE TABLE authors (
   id INTEGER PRIMARY KEY,
   name TEXT
)
```

```
CREATE TABLE books (
   id INTEGER PRIMARY KEY,
   title TEXT,
   authorId INTEGER NOT NULL
     REFERENCES authors(id)
)
```

```
CREATE TABLE books (
  id INTEGER PRIMARY KEY,
  title TEXT,
  authorId INTEGER NOT NULL
    REFERENCES authors(id)
)
```

many-to-many

```
CREATE TABLE authors (
   id INTEGER PRIMARY KEY,
   name TEXT
)
```

```
CREATE TABLE flights (
   id INTEGER PRIMARY KEY,
   airline TEXT
)
```

```
CREATE TABLE tickets (
  id INTEGER PRIMARY KEY,
  authorId INTEGER NOT NULL
    REFERENCES authors(id),
  flightId INTEGER NOT NULL
    REFERENCES flights(id)
)
```

bit.ly/
devmtnsql

SQL Injection

```
db.query(
   'SELECT * FROM injuries' +
   'WHERE name = ' +
   '"' + query.name + '"'
)
```

```
query.name = "Nintendo Thumb"

?query=Nintendo+Thumb
```

```
query.name = "Nintendo Thumb"

SELECT * FROM injuries
WHERE name = 'Nintendo Thumb'
```

```
query.name =
"';DROP TABLE injuries --"
```

```
query.name =
"';DROP TABLE injuries --"

SELECT * FROM injuries
WHERE name = '';
DROP TABLE injuries; --'
```

```
db.query(`
  SELECT * FROM injuries
  WHERE name = $1
`, [query.name])
```

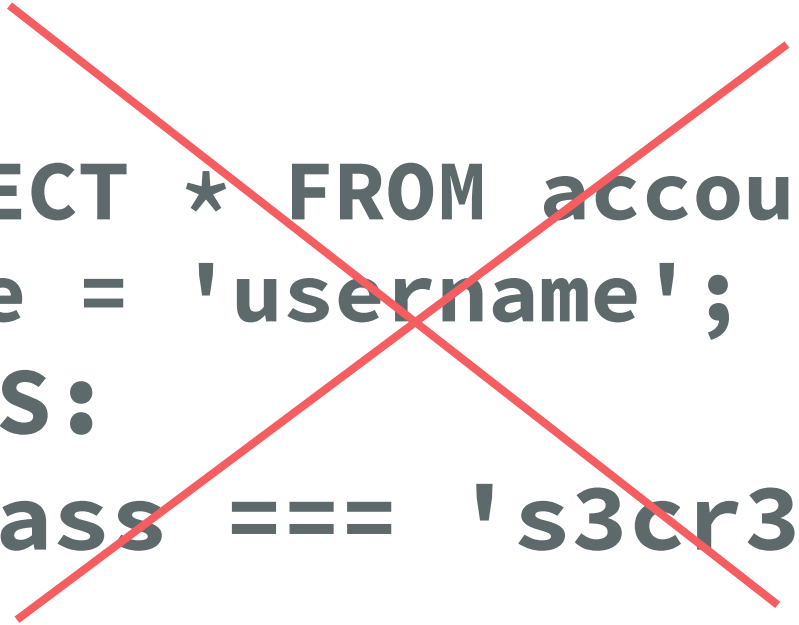
Passwords

```
INSERT INTO account
values('username', 's3cr3t')
```

```
SELECT * FROM account
WHERE name = 'username';

//JS:
a.pass === 's3cr3t'
```

```
INSERT INTO account
values('username', 's3cr3t')
```


SELECT * FROM account WHERE
name = 'username';
//JS:
a.pass === 's3cr3t'

Nefariously Pushing Master

npm

find packages

bcrypt public

build passing dependencies up to date

Lib to help you hash passwords. bcrypt on wikipedia

Catalyst for this module: How To Safely Store A Password

If You Are Submitting Bugs/Issues

First, make sure that the version of node you are using is a *stable* version. You'll know this because it'll have an even major release number. We do not currently support unstable versions and while the module may happen to work on some unstable versions you'll find that we quickly close issues if you're not using a stable version.

If you are on a stable version of node, we can't magically know what you are doing to expose an issue, it is best if you provide a snippet of code or log files if you're having an install issue. This snippet need not include your secret sauce, but it must replicate the issue you are describing. The issues that get closed without resolution tend to be the ones that don't help us help you. Thanks.

```
bcrypt.hash('s3cr3t')

output: 'hash123'
```

```
INSERT INTO account
values('username', 'hash123')
```

```
select * from account where
name = 'username';

//JS:
bcrypt.compare('s3cr3t', a.pass)
```

Transactions

Scenario

User Invitations

users

id	username	email
1	driver33	drv3@gmail.com

invites

id	secretCode
1	abc123
2	xyz345

1. Look up invite
2. Delete invite
3. Create user

1. Look up invite
2. Delete invite
3. Create user

1. Look up invite
2. Create user
3. Delete invite

1. Look up invite
2. Create user
3. Delete invite

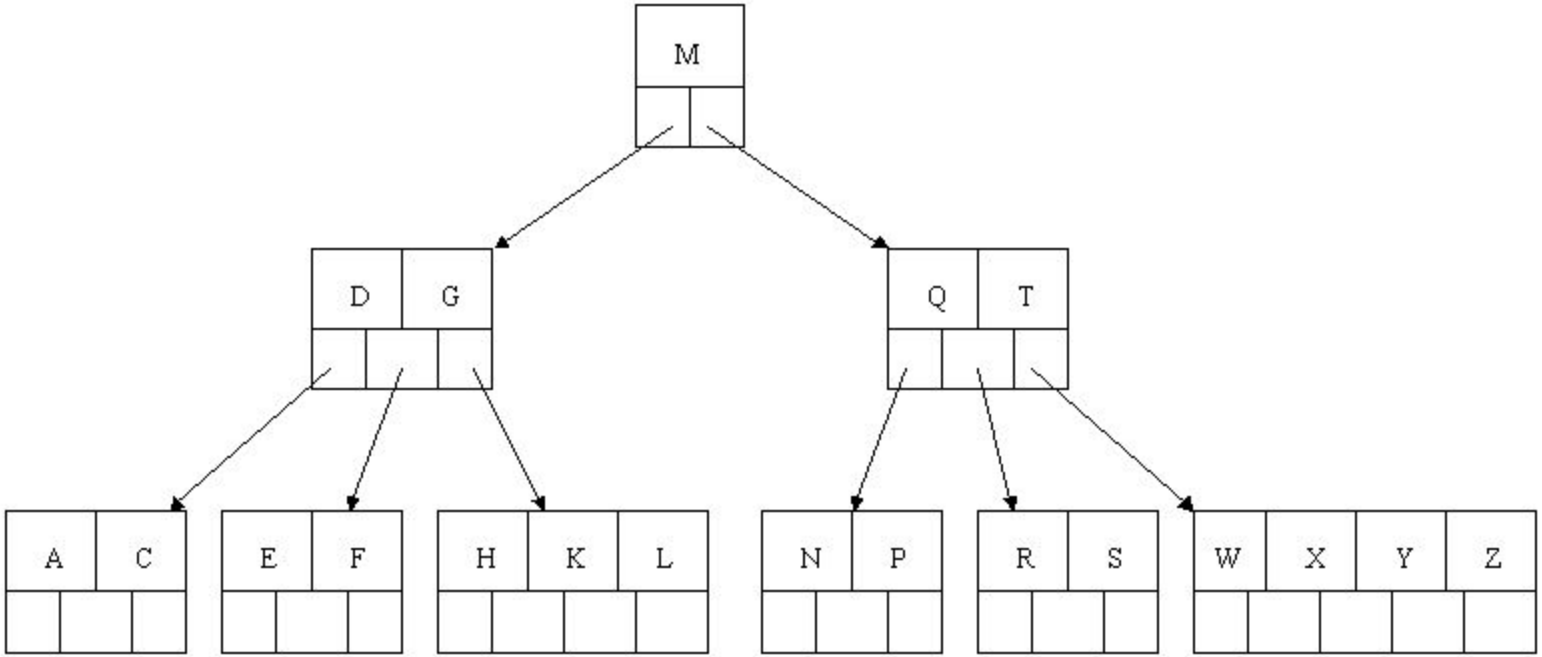
BEGIN

```
SELECT id FROM invites
WHERE secretCode = 'abc123';

BEGIN;
INSERT INTO users…
DELETE FROM invites...
COMMIT;
```

Indexes

Index


M
D
G
Q
T
A
C
E
F
H
K
L
N
P
R
S
W
X
Y
Z

```
CREATE INDEX idx_secretCode
ON invites(secretCode)
```

UNIQUE

```
CREATE UNIQUE INDEX
idx_authorId_flightId
ON tickets(authorId, flightId)
```

FOREIGN KEY

EXPLAIN

```
EXPLAIN QUERY PLAN
SELECT LastName FROM Employee
```

```
EXPLAIN QUERY PLAN
SELECT LastName FROM Employee
WHERE ReportsTo = 4
```

```
EXPLAIN QUERY PLAN
SELECT count(*) FROM Employee
WHERE ReportsTo = 4
```

bit.ly/
massive-demo

bit.ly/
sqlsurvey

Resources

- https://pgexercises.com/
- https://github.com/dhamaniasad/awesome-postgres
- OWASP Top 10 application security risks
  https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
- Object-Role Modeling Fundamentals - Terry Halpin
  https://www.amazon.com/dp/B00VW3BLGQ

@statenjason