

# Databases

Jason Staten

Data

Likes

Stock quotes

# Heartbeats

# Temperature

# Appointments

# Bookmarks



Showtimes

# Earthquakes

Tweets

# Recipes

DNA

GPS coordinates

Data

# Store













***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

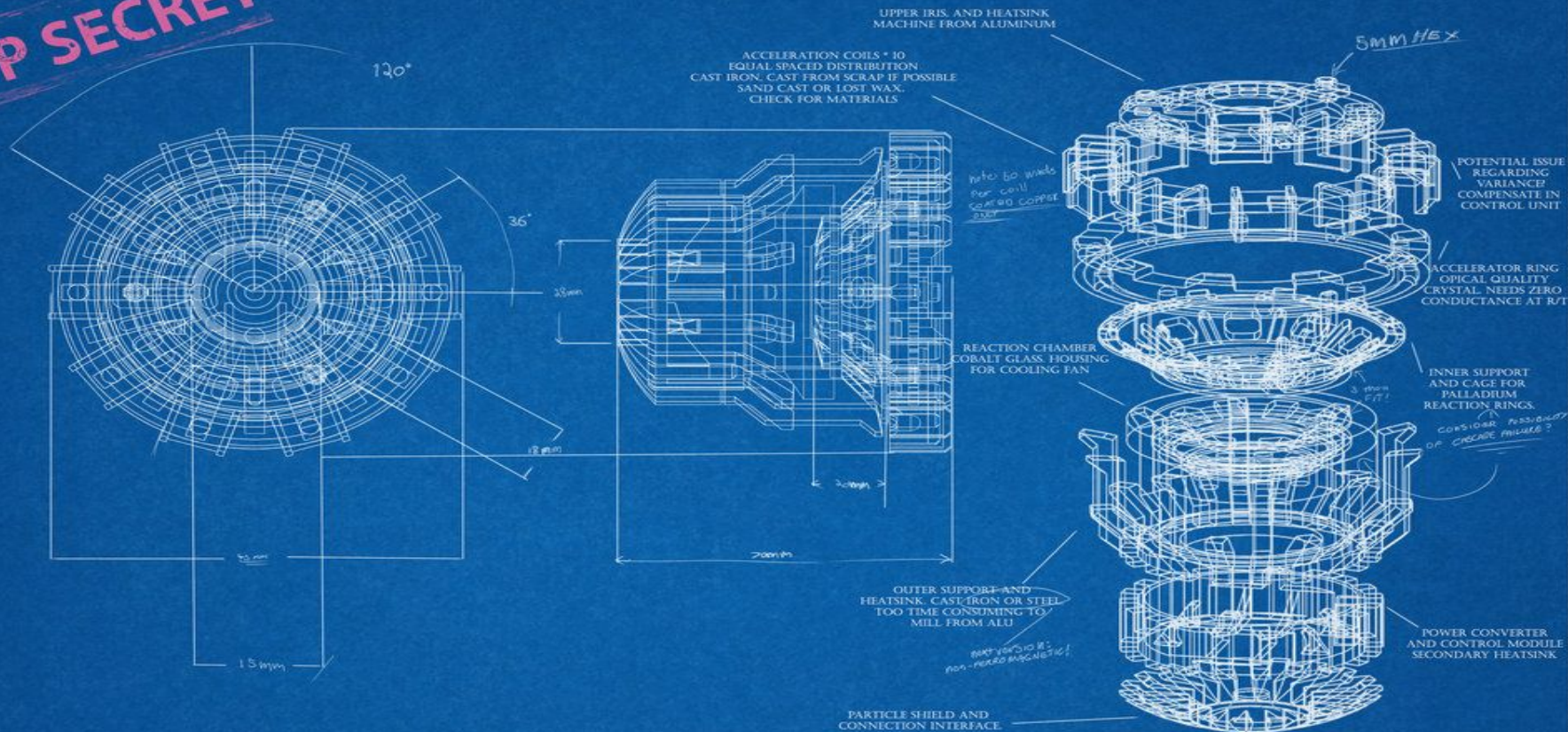
# Tables

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

# Schema



TOP SECRET



STARK INDUSTRIES

SAVANT GUARDE



VERSION: 5.34 (FINAL)  
DESIGNER: [Signature]

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

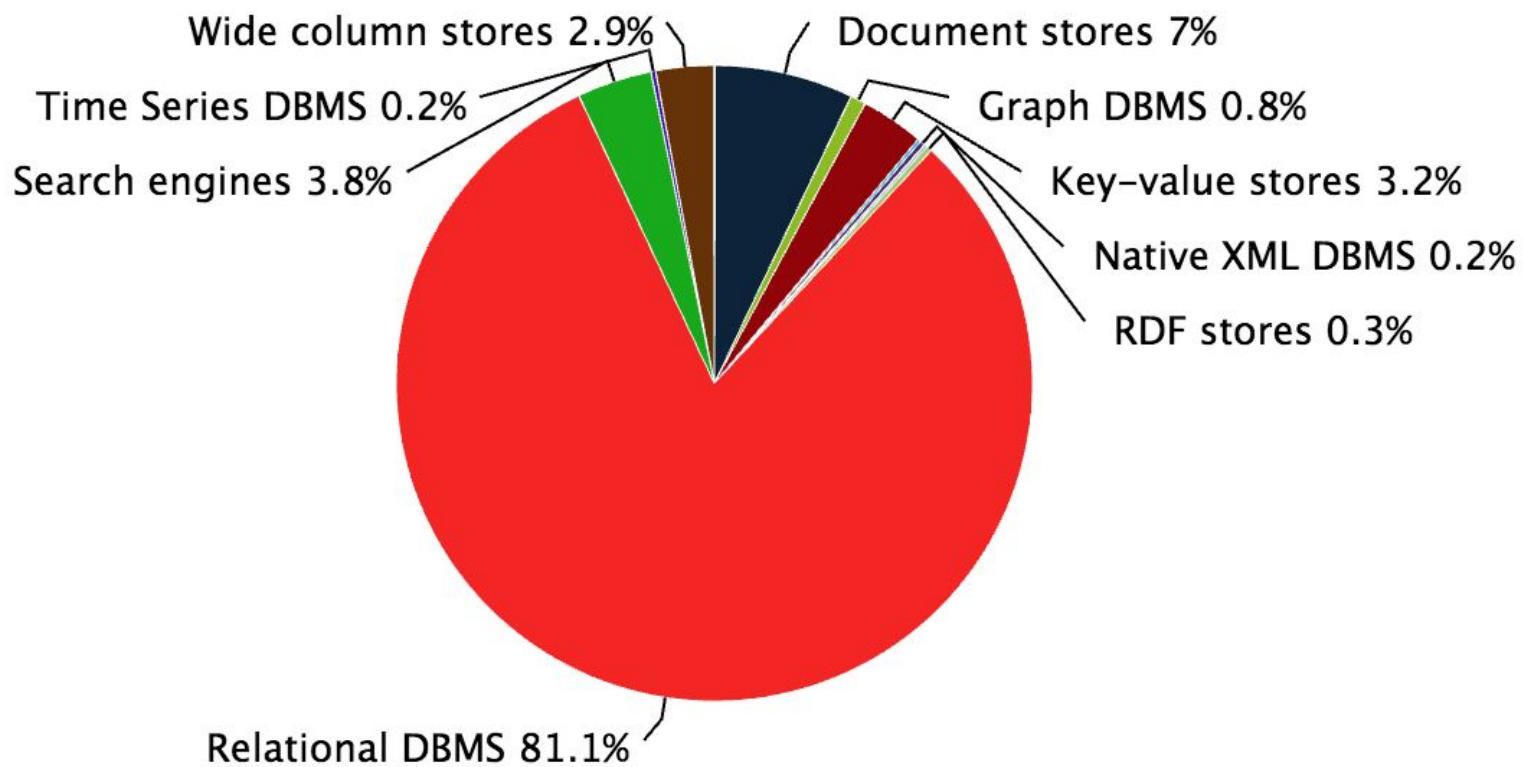
Create tables and rows

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

Artist
ArtistId
Name
Salary



Joined
ArtistId
Name
Salary
AlbumId
Title



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?**

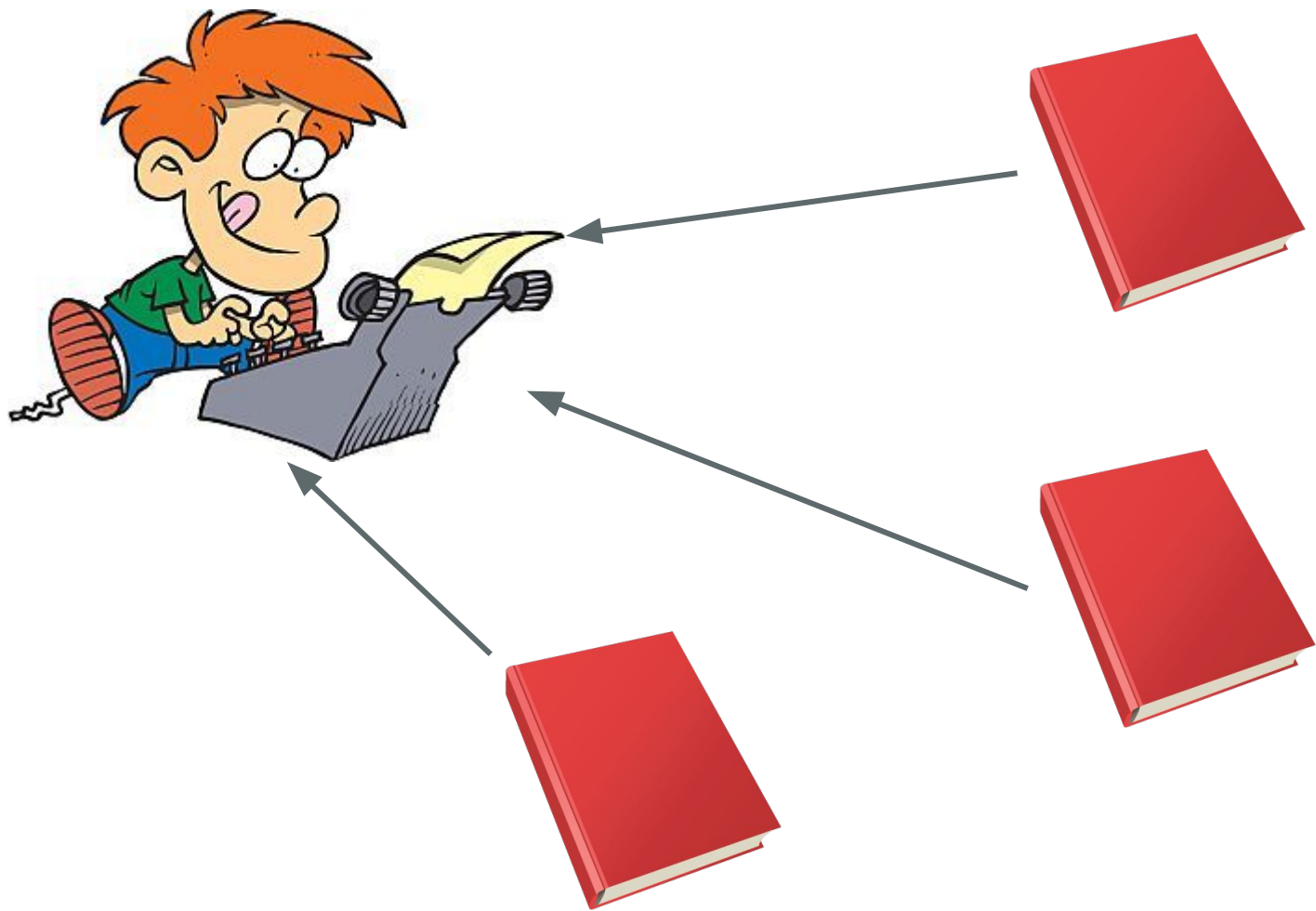
The image features two hands, one from the left and one from the right, reaching towards each other and holding hands. The hands are silhouetted against a bright, warm background of a sunset or sunrise. The sky is filled with soft, glowing light in shades of yellow, orange, and red. The water in the foreground reflects the light from the sky, creating a shimmering effect. The overall mood is romantic and intimate.

# Relationships

many-to-one







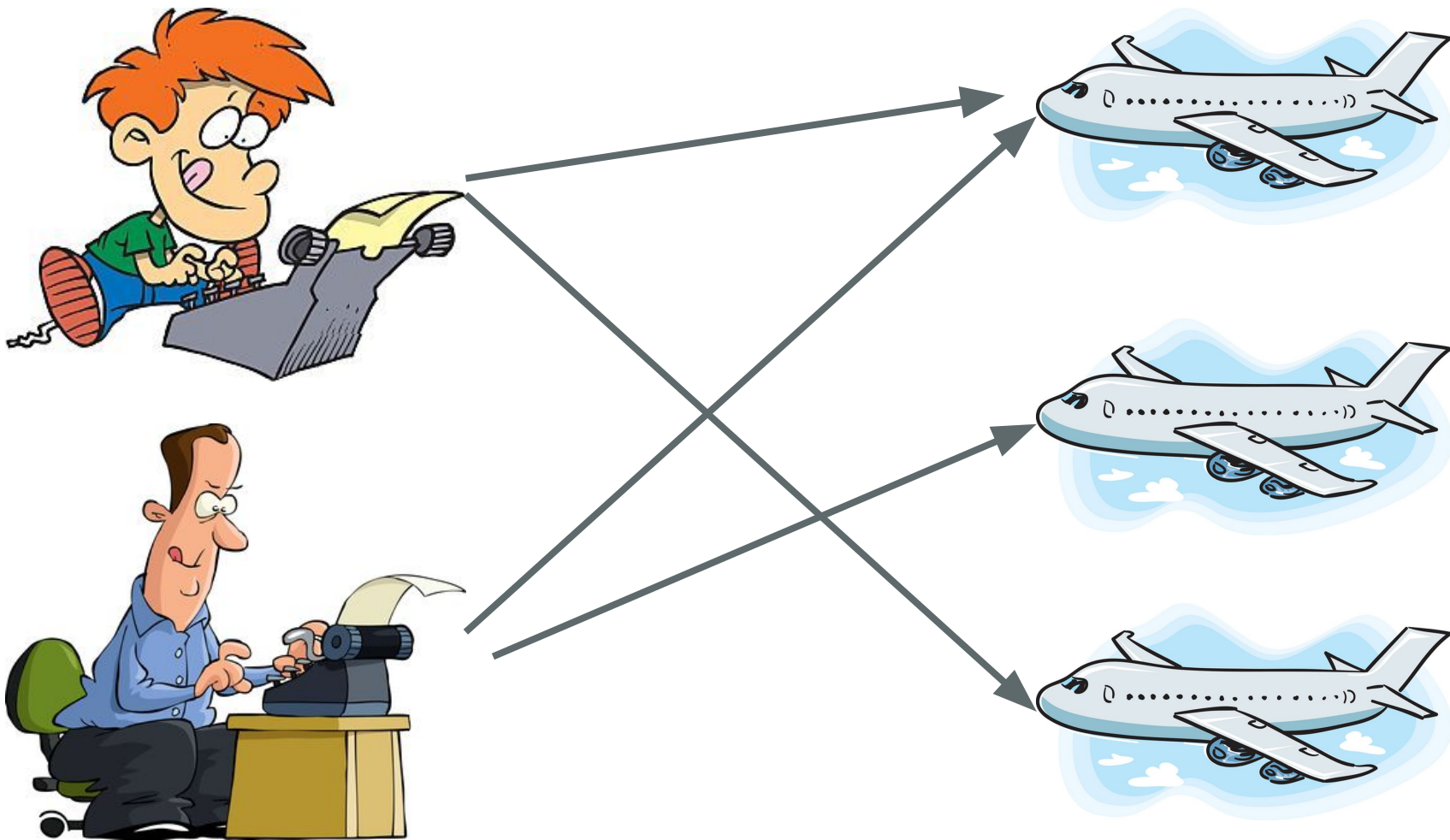
authors

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')
```

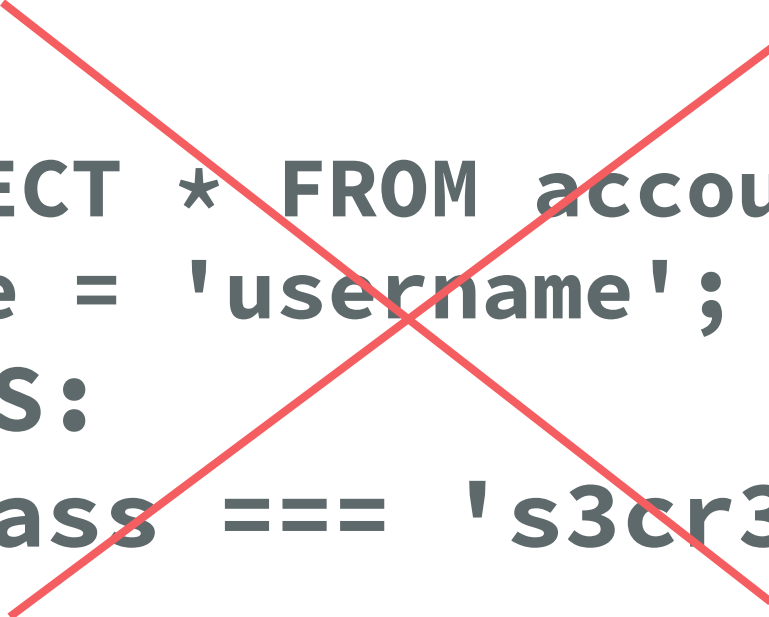
```
SELECT * FROM account WHERE  
name = 'username';
```

```
//JS:
```

```
a.pass === 's3cr3t'
```



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



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



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

## A

### accordion, layouts

- about 128
- movie form, adding 131
- nesting, in tab 128, 129
- toolbar, adding 129-131

### adapters, Ext

- about 18
- using 18, 20

### Adobe AIR 285

### Adobe Integrated Run time. *See* Adobe AIR

### AJAX 12

### Asynchronous JavaScript and XML.

*See* AJAX

## B

### built-in features, Ext

- client-side sorting 86
- column, reordering 86, 87
- columns, hidden 86
- columns, visible 86

### button, toolbars

- creating 63
- handlers 67, 68
- icon buttons 67
- split button 64

### buttons, form 53

## C

### cell renderers

- about 82

### lookup data stores, creating 83

### two columns, combining 84

### classes 254

### ComboBox, form

- about 47
- database-driven 47-50

### component config 59

### config object

- about 28, 29
- new way 28, 29
- old way 28
- tips 26, 29

### content, loading on menu item click 68, 69

### custom class, creating 256-259

### custom component, creating 264-266

### custom events, creating 262-264

## D

### data, filtering

- about 238
- remote, filtering 238-244

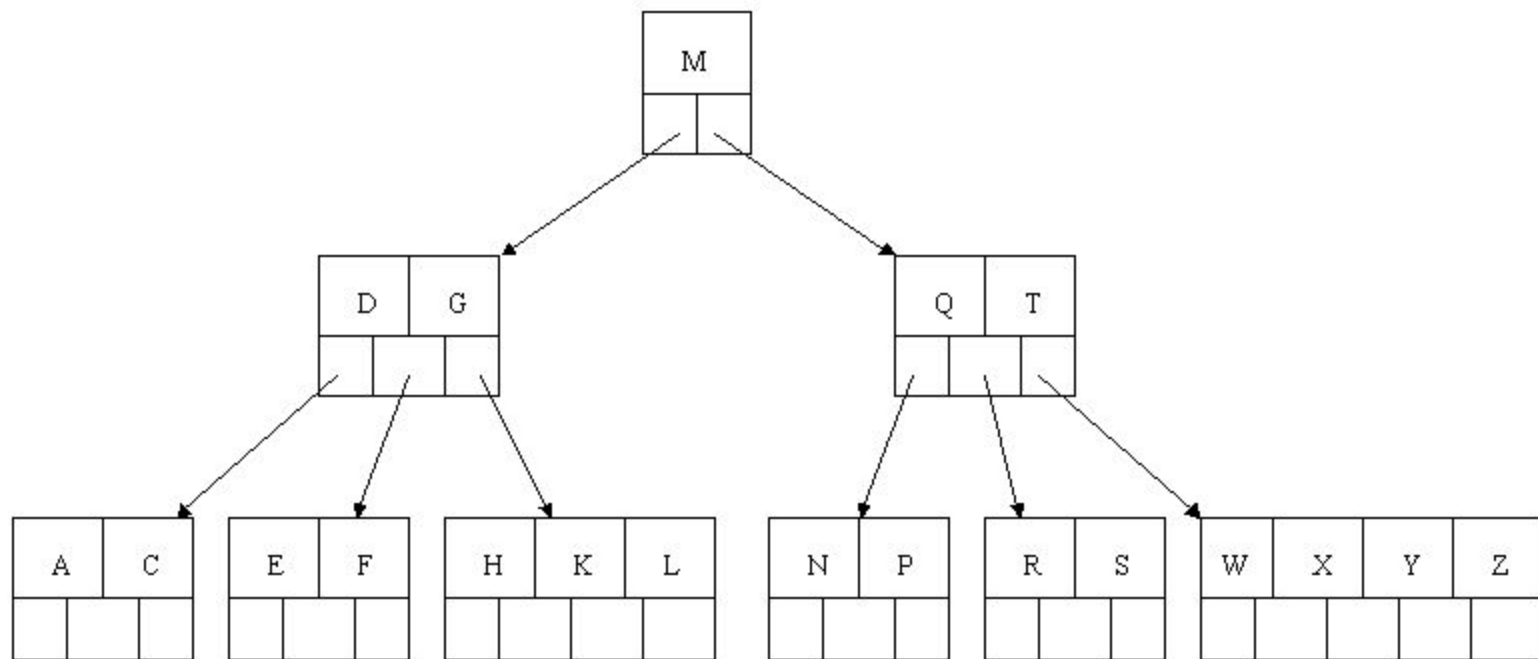
### data, finding

- about 237
- by field value 237
- by record ID 238
- by record index 237

### data, formatting

- about 278
- date, formatting 279
- other formatting 280, 281
- string, formatting 278

### data displaying, GridPanel



```
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**

@statenjason