

Relational DB Design

Last Words on Javascript



Suppose a Database

users				
name	company	company_address	url1	url2
Joe	ABC	1 Work Lane	abc.com	xyz.com
Jill	XYZ	1 Job Street	abc.com	xyz.com

Query to get all info from this database (i.e., userso table)?

```
create table users0 (  
    name varchar(20) NOT NULL,  
    company varchar(20) not null,  
    company_address varchar(20) not null,  
    url1 varchar(20) not null,  
    url2 varchar(20) not null  
);  
insert into users0 (name, company,  
company_address, url1, url2) values ('Joe',  
'ABC', '1 work Lane', 'abc.com', 'xyz.com');  
insert into users0 (name, company,  
company_address, url1, url2) values ('Jill',  
'XYZ', 'i Job Street', 'abc.com', 'xyz.com');
```

DataBase Normalization — 1NF

- First Normal Form — There are no multi-valued attributes
 - Eliminate repeating groups in individual tables.
 - e.g. the url1 & url2 fields
 - Create a separate table for each set of related data.
 - Identify each set of related data with a primary key.
 - So make a “userID” field to indicate that Joe in rows 1 and 2 is the same Joe

```
create table users1 (  
    name varchar(20) NOT NULL,  
    company varchar(20) not null,  
    company_address varchar(20) not null,  
    url varchar(20) not null,  
);  
insert into users1 (name, company, company_address, url)  
values ('Joe', 'ABC', '1 work Lane', 'abc.com');  
insert into users1 (name, company, company_address, url)  
values ('Jill', 'XYZ', 'i Job Street', 'xyz.com');  
insert into users1 (name, company, company_address, url)  
values ('Joe', 'ABC', '1 work Lane', 'abc.com');  
insert into users1 (name, company, company_address, url)  
values ('Jill', 'XYZ', 'i Job Street', 'xyz.com');
```

users				
userId	name	company	company_address	url
1	Joe	ABC	1 Work Lane	abc.com
1	Joe	ABC	1 Work Lane	xyz.com
2	Jill	XYZ	1 Job Street	abc.com
2	Jill	XYZ	1 Job Street	xyz.com

Query to get all info from this database (i.e., users1 table)?

DataBase Normalization — 2NF

- Second Normal Form — Non-key fields must be dependent upon the entire key
 - Create separate tables for sets of values that apply to multiple records.
 - e.g. the URL field for first normal form
 - Relate these tables with a foreign key.
 - reluserid in URLs is a foreign key to userid in users

users			
userid	name	company	company_address
1	Joe	ABC	1 Work Lane
2	Jill	XYZ	1 Job Street

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

```
create table users2 (  
  userid int AUTO_INCREMENT PRIMARY KEY,  
  name varchar(20) NOT NULL,  
  company varchar(20) not null,  
  company_address varchar(20) not null  
);  
insert into users2 (name, company, company_address)  
values ('Joe', 'ABC', '1 work Lane');  
insert into users2 (name, company, company_address)  
values ('Jill', 'XYZ', '1 Job Street');  
  
create table urls2 (  
  urlId int AUTO_INCREMENT PRIMARY KEY,  
  relUserId int,  
  url varchar(20),  
  FOREIGN KEY(relUserId) REFERENCES users2(userid)  
);  
insert into urls2 (reluserid, url)  
values (1, 'abc.com');  
insert into urls2 (reluserid, url)  
values (1, 'xyz.com');  
insert into urls2 (reluserid, url)  
values (2, 'abc.com');  
insert into urls2 (reluserid, url)  
values (2, 'xyz.com');
```

Query to get all info from this database (i.e., users2 and urls2 tables)?

DataBase Normalization — 3NF

- Third Normal Form
 - Eliminate fields that do not depend on the key.
 - The company affiliation of Joe and Jill is NOT an attribute of those people and may be shared

users		
userId	name	relCompld
1	Joe	1
2	Jill	2

companies		
compld	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

```
create table company3 (  
  companyID int AUTO_INCREMENT PRIMARY KEY,  
  company varchar(20),  
  company_address varchar(20)  
);  
insert into company3 (company, company_address)  
values ('ABC', '1 Work Lane');  
insert into company3 (company, company_address)  
values ('XYZ', '1 Job Street');  
create table user3 (  
  userid int AUTO_INCREMENT PRIMARY KEY,  
  name varchar(20) NOT NULL,  
  RelCompanyId int,  
  FOREIGN KEY(RelCompanyId) REFERENCES  
company3(companyID)  
);  
insert into user3 (name, RelCompanyId) values  
( 'Joe', 1);  
insert into user3 (name, RelCompanyId) values  
( 'Jill', 2);  
/** table urls3 is identical to tables urls2 **/
```

Query to get all info from this database (i.e., company3, users3, and urls3 tables)?

DataBase Normalization — 4NF

- The URLs table has info in it that is not an attribute of the URL. Namely the “reluser” column.
- This feels wrong.
 - Clean up by replacing the URL table with two:
 - URLs and URL_relations
- Fourth Normal Form
 - In a many-to-many relationship, independent entities can not be stored in the same table.

```

/** company4 and user4 exactly follow company3 and user3 */
create table urls4 (
    urlid int AUTO_INCREMENT PRIMARY KEY,
    url varchar(20)
);
insert into urls4 (url) values ('abc.com');
insert into urls4 (url) values ('xyz.com');
create table url_rel4 (
    relationID int AUTO_INCREMENT PRIMARY KEY,
    RelURLid int,
    RelCompanyID int,
    FOREIGN KEY(RelURLid) REFERENCES urls4(urlid),
    FOREIGN KEY(RelCompanyID) REFERENCES company4(companyID)
);
insert into url_rel4 (RelURLid, relcompanyid) values (1, 1);
insert into url_rel4 (RelURLid, relcompanyid) values (1, 2);
insert into url_rel4 (RelURLid, relcompanyid) values (2, 1);
insert into url_rel4 (RelURLid, relcompanyid) values (2, 2);
    
```

users		
userid	name	relCompId
1	Joe	1
2	Jill	2

urls	
urlid	url
1	abc.com
2	xyz.com

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

url_relations		
relationId	relatedUrlId	relatedUserId
1	1	1
2	1	2
3	2	1
4	2	2

Inserts into normalized DBs

- Auto increments id fields are really handy BUT
 - how do you know what they are when you insert into another table?
 - E.g.,

```
insert into user3 (name, RelCompanyId) values ('Joe', 1);
insert into user3 (name, RelCompanyId) values ('Jill', 2);
```
 - Need to do insert that uses a query!
 - Simplest

```
INSERT INTO tableC (c1, c2, c3) SELECT dd1, dd2, dd3 FROM tableDD;
```

 - This violates RDB principle of only storing data once!
 - More useful
 - ```
insert into user4 (name, RelCompanyId) select 'Joe', company4.companyid
from company4 where company4.company='ABC';
```

# Querying this database

| users  |      |           |
|--------|------|-----------|
| userId | name | relCompId |
| 1      | Joe  | 1         |
| 2      | Jill | 2         |

| companies |         |                 |
|-----------|---------|-----------------|
| compId    | company | company_address |
| 1         | ABC     | 1 Work Lane     |
| 2         | XYZ     | 1 Job Street    |

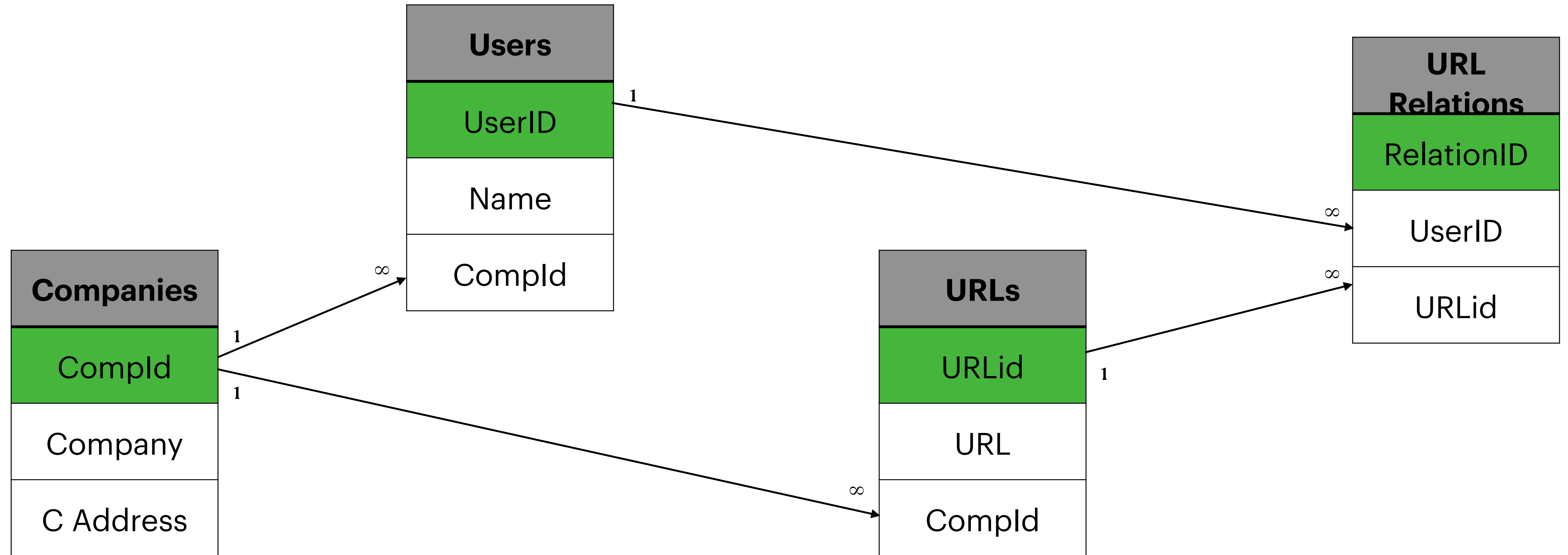
| urls  |         |
|-------|---------|
| urlId | url     |
| 1     | abc.com |
| 2     | xyz.com |

| url_relations |              |               |
|---------------|--------------|---------------|
| relationId    | relatedUrlId | relatedUserId |
| 1             | 1            | 1             |
| 2             | 1            | 2             |
| 3             | 2            | 1             |
| 4             | 2            | 2             |

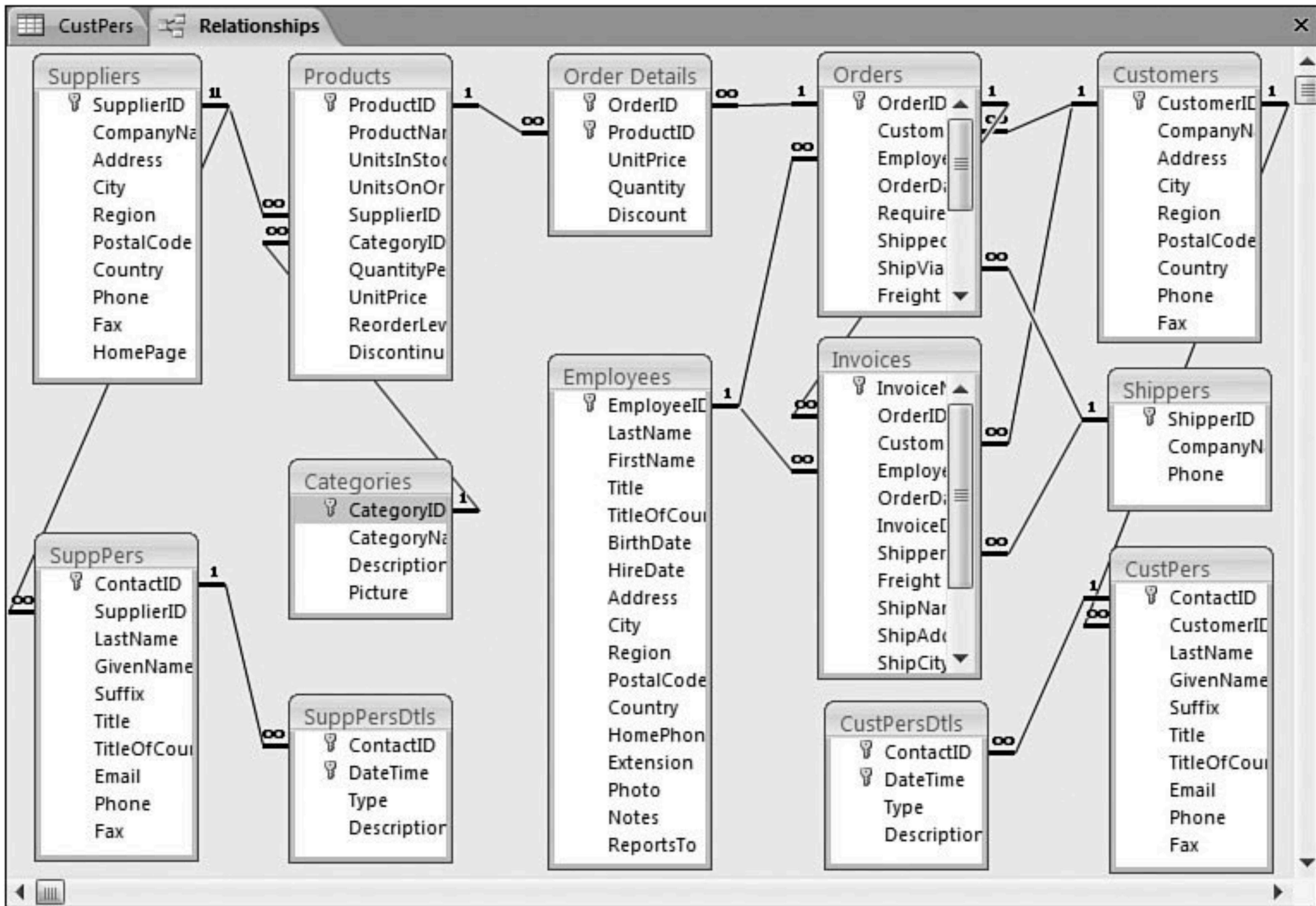
- Insert for URL\_RELATIONS table
- List all of Joe's urls
- List all employees of company 1
- List all employees of the company that Joe works for
- How many companies are in the database?



# Representing a database



- Table Name at the top (grey background)
- Primary key next (green background)
- Arrows indicate foreign keys
- Numbers by arrows indicate type of relationship

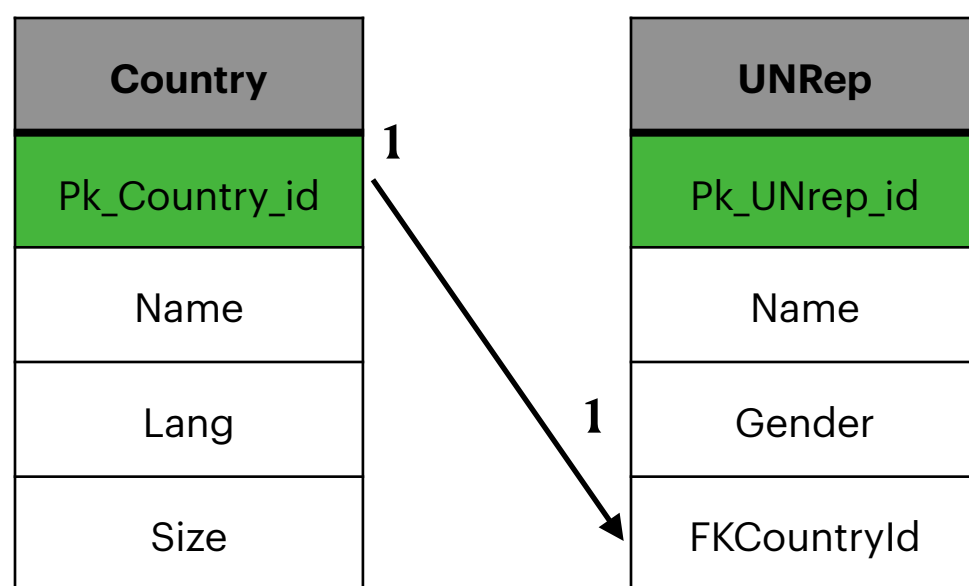


# Keys relation in enforced by DB

- One-to-One

```
CREATE TABLE Country
(
Pk_Country_Id INT PRIMARY
KEY,
Name VARCHAR(100),
Officiallang VARCHAR(100),
Size INT(11)
);
```

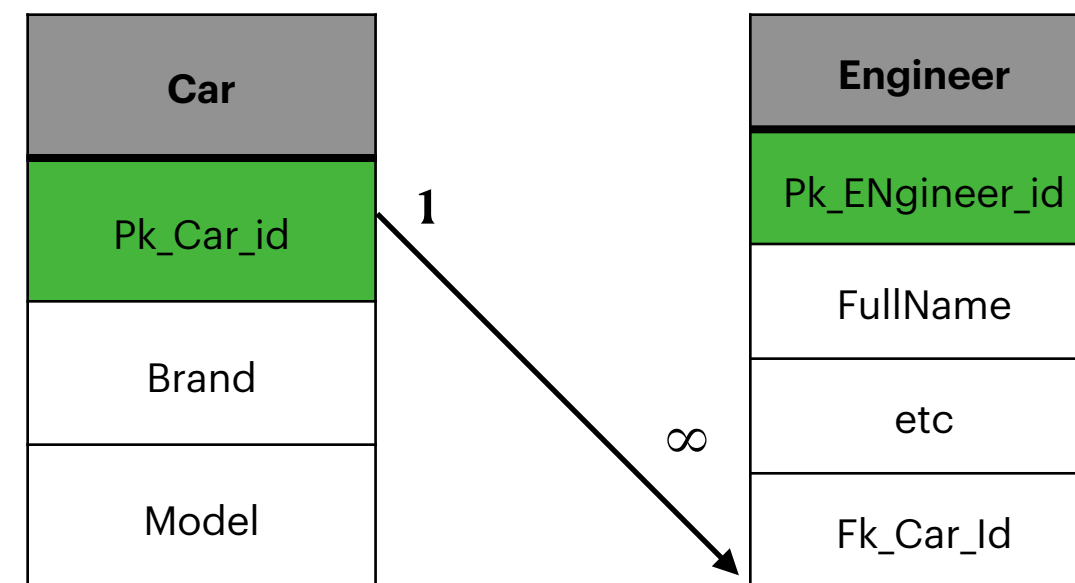
```
CREATE TABLE UNrep
(
Pk_UNrep_Id INT PRIMARY KEY,
Name VARCHAR(100),
Gender VARCHAR(100),
Fk_Country_Id INT UNIQUE,
FOREIGN KEY(Fk_Country_Id)
REFERENCES
Country(Pk_Country_Id)
);
```



- One-to-Many

```
CREATE TABLE Car
(
Pk_Car_Id INT PRIMARY KEY,
Brand VARCHAR(100),
Model VARCHAR(100)
);
```

```
CREATE TABLE Engineer
(
Pk_Engineer_Id INT PRIMARY
KEY,
FullName VARCHAR(100),
MobileNo CHAR(11),
Fk_Car_Id INT,
FOREIGN KEY(Fk_Car_Id)
REFERENCES Car(Pk_Car_Id)
);
```

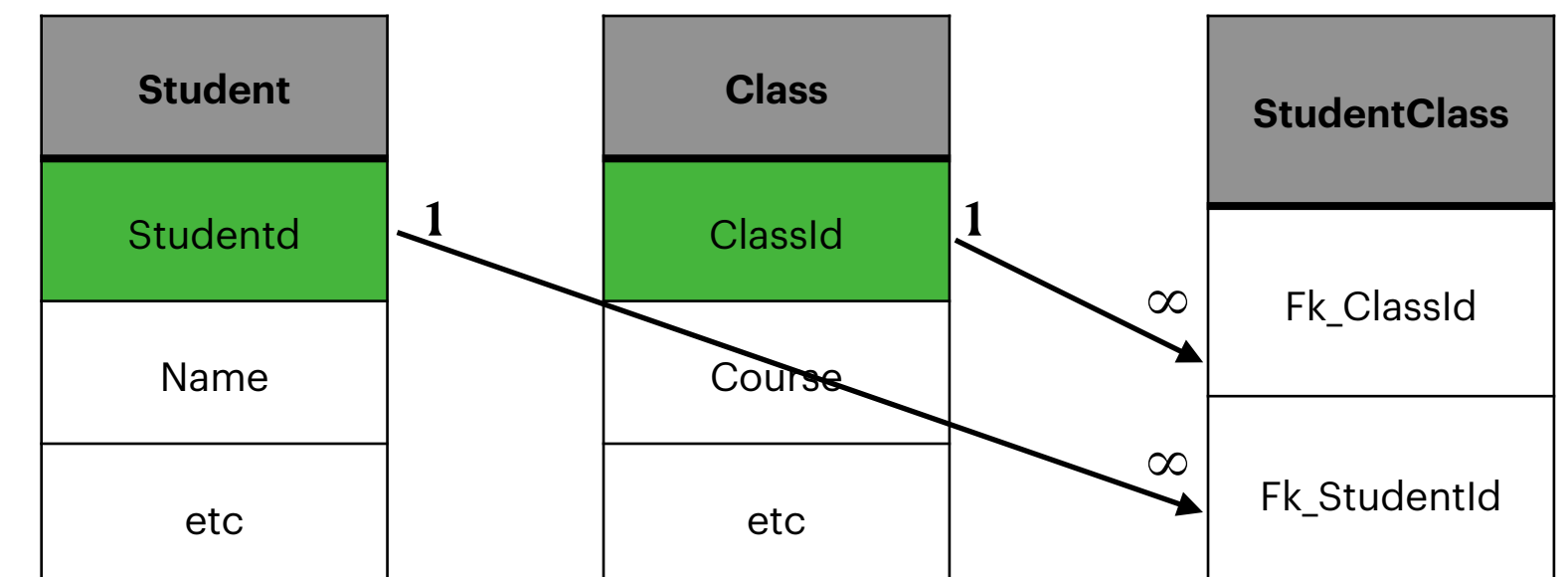


- Many-to-Many

```
CREATE TABLE Student(
StudentID INT(10),
Name VARCHAR(100),
PRIMARY KEY(studentID)
);
```

```
CREATE TABLE Class(
ClassID INT(10) PRIMARY KEY,
Course VARCHAR(100)
);
```

```
CREATE TABLE StudentClassRelation(
Fk_StudentID INT(15) NOT NULL,
Fk_ClassID INT(14) NOT NULL,
FOREIGN KEY (Fk_StudentID) REFERENCES
Student(StudentID),
FOREIGN KEY (Fk_ClassID) REFERENCES
Class(ClassID),
UNIQUE (StudentID, ClassID)
);
```



Use 2 1-∞ relations to create a ∞-∞ relation

# Database creation tips

- Keep data items atomic (e.g., first and last names are separate). Concatenating columns together later on-the-fly is generally easy, but separating them is not. (First Normal Form)
- Define the primary key first. Use a descriptive name (PARCELID, not ID)
- Use a single column for the primary key whenever possible; multi-column primary keys are appropriate for many-to-many relationships
- Use lookup tables rather than storing long values
  - Lookup tables are tables that have a key and a name ... for instance: car manufacturers, car models, ... They tend to be static
- Use numeric keys whenever possible (What about ZIP codes?)
- Avoid using multiple columns to represent a one-to-many relationship (e.g., columns such as CHILD<sub>1</sub>, CHILD<sub>2</sub> in a table called PARENT rather than putting the children in a separate table. (First Normal Form)
- For readability, use the primary key name for foreign keys unless the same foreign key is used multiple times in the same table (e.g., state of work and state of residence for a person might both be foreign keys that reference a table of states)
- Do not include two columns whose values are linked together (e.g., county name and county ID) unless one of the columns is the primary key of the table (Third Normal Form)
- Avoid allowing NULL values in columns that have a discrete range of possible values (e.g., integers between 1 and 10, inclusive)
- Avoid using multiple tables with similar structures that represent minor variants on the same entity (e.g., putting Boston parcels and Cambridge parcels in separate tables).

# My JS Game

## Concentration

- Idea, just match up pairs
- But, only show pairs briefly if not a match
- Desires:
  - Multiple game levels
  - Leaderboards
  - Reproducible game boards
    - makes for easier debugging

# Concentration animation

- General idea:
  - roll up a window shade to see the secret.
  - If secrets match Roll up another window shade.
    - then roll up a further shade for reveal something indicating success.
    - else, wait a while and roll down both shades
  - Each shade implemented with a div. So each item is actually a stack of 3 divs
- Problem when clicking quickly
  - [Lec15/cb.html](#)
  - Rather than trying to understand in the game
    - [Lec15/scopeTimeout.html](#)

# Concentration — Images

- Just uses standard html emojis
- Problem — How to make these as big as possible but still fit.
  - Images would be easier — just set the width/height to the fit in space
- But, what font size is “as big as possible”?
  - For simplicity assume that width is the constraining variable.
  - Further assume that you only want to show one character
    - Could handle both width and height, just more code.
    - let viewWid ← width of the “viewport” (that is, the width of the entire browser window)
    - let cardWid ← width of one concentration card
    - let pct ←  $100 * \text{cardWid} / \text{viewWid}$
  - then in css: { 'font-size': pct+'vw' }
    - 'vw' is “viewportwidth”.
      - 100vw would make a font with a single char the width of the screen

# Concentration — Server Side

- Game levels stored on server.
  - This means that client must be flexible about buttons — or dropdowns — to select level.
  - By having a discrete set of levels leaderboards are easier.
- Scores stored in a single table

| difficulty | description | width | height | seeder          |
|------------|-------------|-------|--------|-----------------|
| 1          | Easy        | 2     | 2      | aklsjdlsdkfjsd8 |
| 2          | Medium      | 2     | 4      | aklsjdkter54    |
| 3          | Harder      | 4     | 4      | aklsjdsads      |
| 4          | Hard        | 8     | 4      | aklsjdkasdas    |

| Field        | Type        | Null | Key | Default             | Extra      |
|--------------|-------------|------|-----|---------------------|------------|
| difficulty   | int(11)     | NO   | MUL | NULL                |            |
| recnum       | int(11)     | NO   | PRI | NULL                | auto_incre |
| scorer       | varchar(64) | YES  |     | NULL                |            |
| score        | int(11)     | YES  |     | NULL                |            |
| DateRecorded | datetime    | YES  |     | current_timestamp() |            |

| difficulty | recnum | scorer | score | DateRecorded        |
|------------|--------|--------|-------|---------------------|
| 1          | 2      |        | 6     | 2020-09-11 22:13:57 |
| 1          | 3      | testt  | 5     | 2020-09-11 22:38:32 |
| 1          | 4      |        | 6     | 2020-11-01 08:27:43 |
| 1          | 5      |        | 4     | 2020-11-01 08:49:17 |



# Music Radio Station

- To decide what music to play next need to know
  - Music in library
  - Music played recently
  - Customer requests
  - Ratings