Database Schema Design

Organizing Data

Databases


I THINK WE SHOULD BUILD AN SQL DATABASE.
UH-OH
DOES HE UNDERSTAND WHAT HE SAID OR IS IT SOMETHING HE SAW IN A TRADE MAGAZINE AD?
WHAT COLOR DO YOU WANT THAT DATABASE?
I THINK MAUVE HAS THE MOST RAM.
E-mail: SCOTTADAMS@AOL.COM
© 1996 United Feature Syndicate, Inc.(NYC)

Why you care

SQL is the gateway to the world of Databases

Reading & Writing Raw SQL is a skill that will set you apart from other Devs

Super Important for implementing & debugging advanced Database systems

Database Schema Design

Good Database Design should:

- Capture state, not behavior
- Design relationships between tables

State and Behavior

```
class OrangeTree
  def initialize
    @age = 5
    @height = 5
    @oranges = produce_oranges
  end

  def produce_oranges
    Array.new(3) { Orange.new }
  end
end
```

```
class Orange
  def initialize
    @diameter = rand(2..4)
  end
end
```

State and Behavior

```
orange_tree = OrangeTree.new

⇒ #<OrangeTree 0x007fe1f1a4fb20
     @age=5,
     @height=5,
     @oranges=[#<Orange:0x007fe1f1a4faa8 @diameter=2>,
               #<Orange:0x007fe1f1a4fa80 @diameter=4>,
               #<Orange:0x007fe1f1a4fa58 @diameter=3>]>
```

Temporary State

- We create objects to hold data
- Data is lost when our program has finished executing

☹

Temporary State Problems

- What if we want to use the current state later?

Temporary State Problems

- What if we want to use the current state later?

  Mega Man vs. Mega Man II

  Phone number from a super hot crush

  50 million client purchase records

Persistant State

- To keep current state we need to save the data

Persistant State

Where to save it?

CSV

Pen && Paper

☺Database☺

Databases

- Like object-oriented programming, databases model the data in real world systems

Databases and Ruby

- How do objects in databases relate to Ruby?

Databases and Ruby

Modeling State	
Ruby	***Database***
Classes	
Instances of classes	
Instance variables	

Databases and Ruby

Modeling State	
Ruby	***Database***
Classes	***Tables***
Instances of classes	
Instance variables	

Databases and Ruby

Modeling State	
Ruby	***Database***
Classes	***Tables***
Instances of classes	***Rows***
Instance variables	

Databases and Ruby

Modeling State	
Ruby	***Database***
Classes	***Tables***
Instances of classes	***Rows***
Instance variables	***Fields***

AKA: Schema

Schema Design

```
orange_tree = OrangeTree.new

⇒ #<OrangeTree 0x007fe1f1a4fb20
     @age=5,
     @height=5,
     @oranges=[#<Orange:0x007fe1f1a4faa8 @diameter=2>,
               #<Orange:0x007fe1f1a4fa80 @diameter=4>,
               #<Orange:0x007fe1f1a4fa58 @diameter=3>]>
```

Schema Design

orange_trees
id
created_at
updated_at

oranges
id
created_at
updated_at

Schema Design

orange_trees
id
created_at
updated_at

oranges
id
created_at
updated_at

Schema Design

orange_trees
id
age
height
created_at
updated_at

oranges
id
created_at
updated_at

Schema Design

orange_trees
id
age
height
created_at
updated_at

oranges
id
created_at
updated_at

Schema Design

orange_trees
id
age
height
created_at
updated_at

oranges
id
diameter
created_at
updated_at

Schema Design

orange_trees
id
age
height
name
created_at
updated_at

oranges
id
diameter
created_at
updated_at

Schema Design

- Schema is more than just static tables
- Relationships are kinda of a big deal

Types of Relationships:

- One to One
- One to Many
- Many to Many

Schema Design

- What is the relationship between orange trees and oranges?

Schema Design

- What is the relationship between orange trees and oranges?

  An **orange tree** has many **oranges**.

Schema Design

- What is the relationship between orange trees and oranges?

  An **orange tree** has many **oranges**.
  An **orange** belongs to an **orange tree**.

DB Relationships: Talk it out.

- Relationships are two way streets
- Should sound like a natural sentence

An **orange tree** has many **oranges**.
An **orange** belongs to an **orange tree**.

Schema Design

- How to link between tables?

Schema Design

- How to link between tables?

Primary key: unique identifier table field

Every table has autogenerated Primary Keys, unless specifically disabled

Foreign key: another table's unique identifier

Only tables which belong to another table contain a Foreign Key

Schema Design

orange_trees
id
age
height
created_at
updated_at

oranges
id
diameter
created_at
updated_at

- On which table does the foreign key belong?

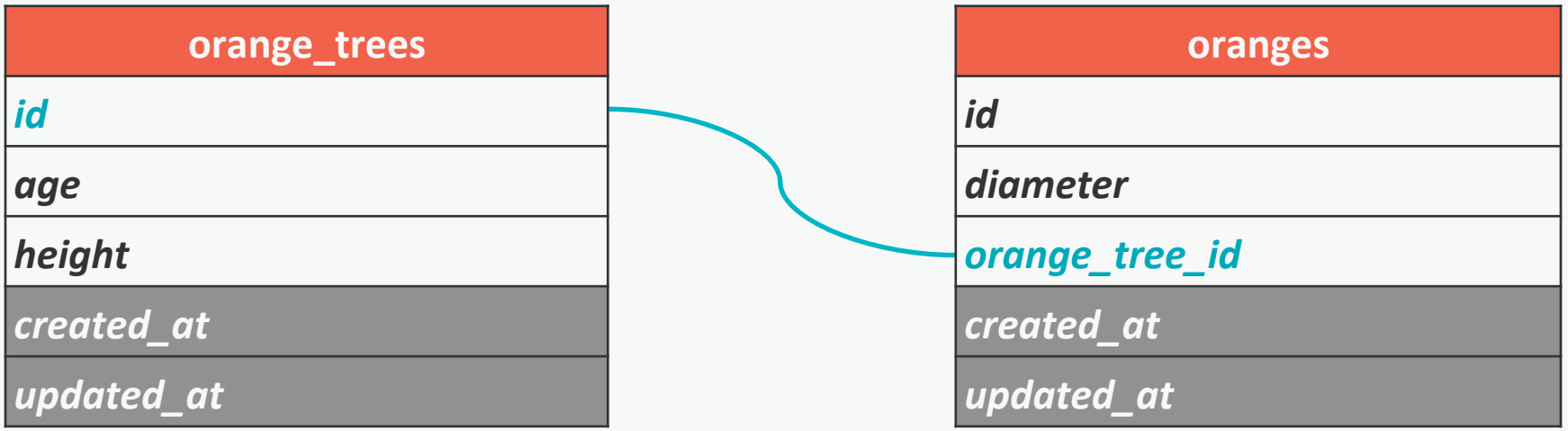
Schema Design

orange_trees
id
age
height
created_at
updated_at

oranges
id
diameter
orange_tree_id
created_at
updated_at

- The foreign key belongs on the table of the objects that are owned

Schema Design



- The foreign key belongs on the table of the objects that are owned

Schema Design

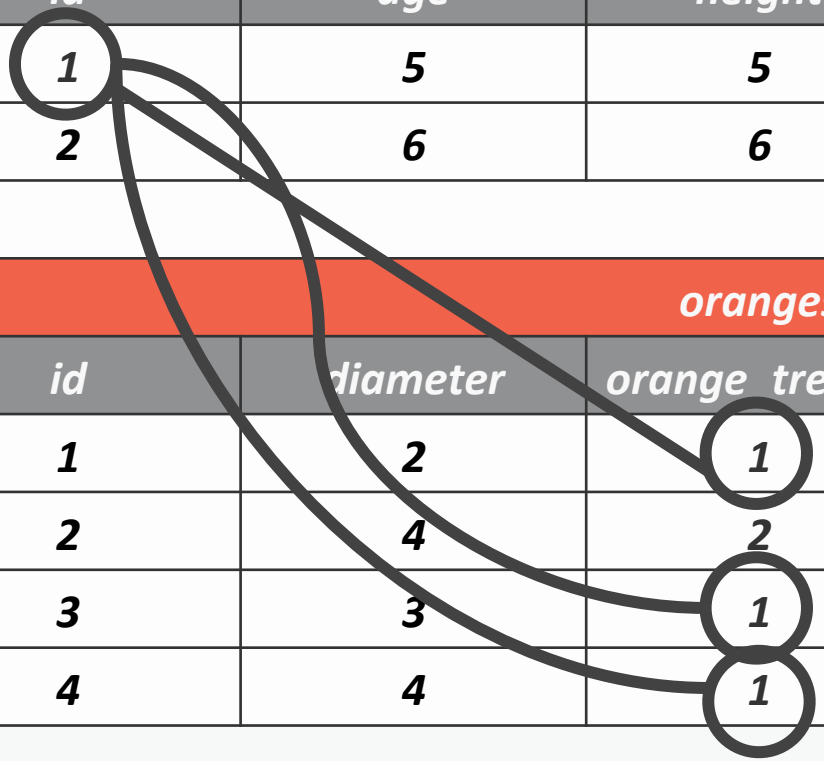
orange_trees				
id	age	height	created_at	updated_at
1	5	5	2014-03-22	2014-03-22
2	6	6	2014-03-22	2014-03-22

oranges				
id	diameter	orange_tree_id	created_at	updated_at
1	2	1	2014-03-22	2014-03-22
2	4	2	2014-03-22	2014-03-22
3	3	1	2014-03-22	2014-03-22
4	4	1	2014-03-22	2014-03-22

Schema Design

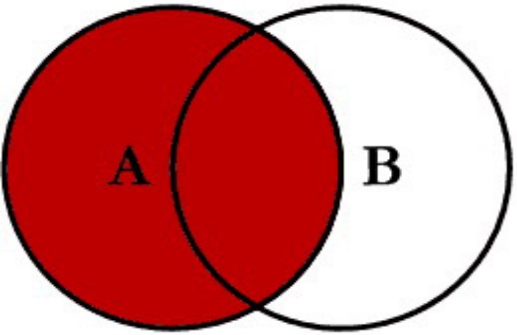
orange_trees				
id	age	height	created_at	updated_at
1	5	5	2014-03-22	2014-03-22
2	6	6	2014-03-22	2014-03-22

oranges				
id	diameter	orange_tree_id	created_at	updated_at
1	2	1	2014-03-22	2014-03-22
2	4	2	2014-03-22	2014-03-22
3	3	1	2014-03-22	2014-03-22
4	4	1	2014-03-22	2014-03-22

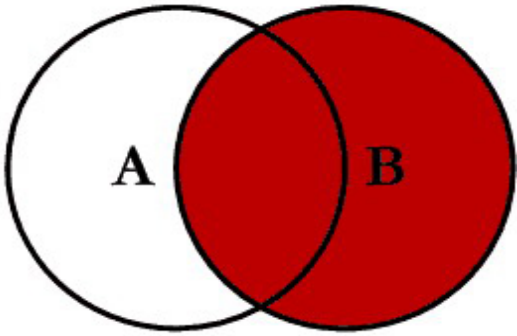


SQL JOINS

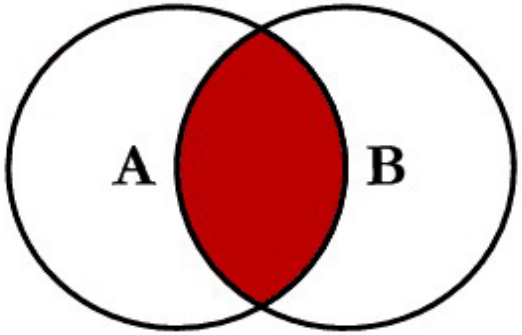
More Detail @ the Source



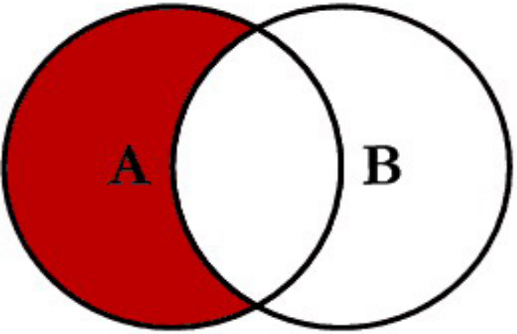

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



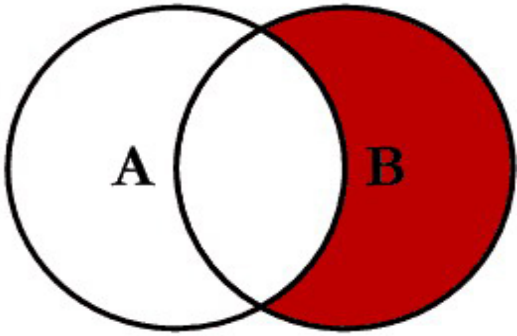

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



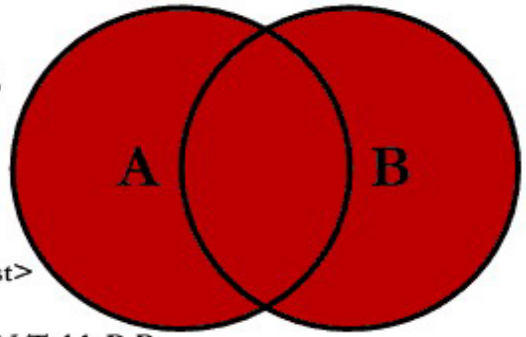

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



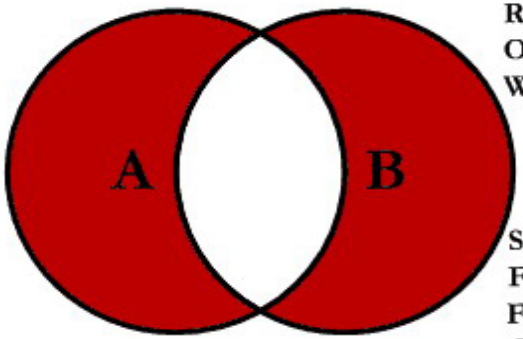

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

SQL Statements

orange_trees
id
age
height
name
created_at
updated_at

```
CREATE TABLE orange_trees
(
id integer PRIMARY KEY AUTOINCREMENT,
age int NOT NULL,
height int NOT NULL,
name varchar(255),
created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

SQL Statements

oranges
id
diameter
created_at
updated_at

```
CREATE TABLE oranges
(id integer PRIMARY KEY,
diameter integer NOT NULL,
created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

SQL Statements

```
INSERT INTO orange_trees
(age, height, name)
VALUES
(1,10,'Brick'),
(1,15,'Bill'),
(2,22,'Jordan');

INSERT INTO oranges(diameter)
VALUES(3);
```

SQL Statements

```
SELECT name FROM orange_trees
WHERE age = 1 AND height < 20;

SELECT * FROM oranges
WHERE diameter = 3;
```

SQL Links:

http://www.cheat-sheets.org/sites/sql.su/

https://dev.mysql.com/doc/refman/5.0/en/group-by-functions.html

http://lmgtfy.com/?q=SQL+group+by+aggregate+functions

SQL CHEET SHEET

SQL SELECT STATEMENTS

```
SELECT * FROM t
SELECT c1,c2 FROM t
SELECT c1,c2 FROM t
WHERE  conditions
SELECT c1,c2 FROM t
WHERE conditions
ORDER BY c1 ASC,c2 DESC
SELECT DISTICT c1,c2
FROM t
SELECT c1, aggregate(c2 * c3)
FROM t
GROUP BY c1
SELECT c1, aggregate(c2 * c3)
FROM t
GROUP BY c1
HAVING c1 > v1
```

SQL UPDATE DATABASE

```
INSERT INTO t (c1,c2...)
VALUES (v1,v2...)
INSERT INTO t1 (c1,c2...)
SELECT c1,c2... FROM t2
WHERE conditions
UPDATE t
SET c1 = v1, c2 = v2,...
WHERE conditions
DELETE FROM t
WHERE conditions
TRUNCATE TABLE t
```

SQL OPERATORS

```
SELECT * FROM t
WHERE c1 [NOT] BETWEEN v1
AND v2
SELECT * FROM t
WHERE c1 [NOT ] IN (v1,v2,...)
SELECT* FROM t
WHERE c1 > v1 AND c1 < v2
SELECT * FROM t
WHERE c1 < v1 OR c1 > v2
SELECT * FROM t
WHERE c1 = v1
SELECT * FROM t
WHERE c1 <> v1
```

SQL TABLE STATEMENTS

```
CREATE TABLE t(
c1 dt1(l1),
c2 dt2(l2),
...
)
DROP TABLE t
ALTER TABLE t
ADD COLUMN c dt(l)
ALTER TABLE t
DROP COLUMN c
```

SQL VIEW STATEMENTS

```
CREATE UNIQUE
INDEX idx ON t(c1,c2..)
DROP INDEX t.idx
```

SQL JOIN STATEMENTS

```
SELECT * FROM t1
INNER JOIN t2 ON conditions
SELECT * FROM t
WHERE c1 [NOT ] IN (v1,v2,...)
SELECT * FROM t1
INNER JOIN t2 ON conditions
WHERE conditions
SELECT *
FROM t1, t2
WHERE conditions
SELECT * FROM t1
LEFT JOIN t2 ON conditions
SELECT * FROM t1
RIGHT JOIN t2 ON conditions
SELECT * FROM t1
FULL OUTER JOIN t2 ON conditions
SELECT * FROM t1 AS at1
INNER JOIN t2 AS at2 ON at1.c1 =
at2.c2
```

SQL VIEW STATEMENTS

```
CREATE VIEW vw
AS
SELECT c1,c2
FROM t
ALTER VIEW vw
AS
SELECT c1,c2
FROM t
DROP VIEW vw
```

t:table name, c: column name,vw: view name, v: value,dt: data type,l: data type length, at: table alias,aggregate: aggreate function,idx: index name

SQL && Schema

Questions