Mike Farr - SF-Dragonflies-2014
mikefarr@mac.com
michaelmfarr@gmail.com

Relational: the secret story

There are two types of tables in relational databases:

Type 1) has rows of data in which at least one item is unique, e.g. in a table of bands, each band is unique: there is only one "Rolling Stones". In the database of bands each band has an ID number. The database stores the same information for each band. Likewise, the same is true for a table of musicians.

ID	band-name	most-popular-album	label	ASCAP-number
1	Beatles	Abby Road	Apple	4555-15
2	Yes	Close to the Edge	Virgin	1234-51
3	Plastic Ono	?	Apple	1254-95
4	Paul McCartney Band	Band on the Run	Apple	5747-48

ID	name	status
1	John Lennon	deceased
2	Paul McCartney	Still Kicking

Type 2) is a relationship or "junction" or "join" table (not to be confused with the SQL command JOIN). It relates unique items in one type 1 table to unique items in another type 1 table. For example, each band has several musicians and in some cases musicians may belong to more than one band.

musician_ID	band_ID

1	1
1	3
2	1
2	4

These two types of tables are surprisingly all that's necessary for data to be organized and searched and combined using SQL.

Schemas- how to organize data into tables

Data is organized based on three fundamental relationships. If you can recognize these you can figure out how many tables you need and how to define them:

1 to 1:

A musician has a 1 to 1 relationship to his/her *first_name*, *last_name*, *age*, perhaps a *main_instrument*. A band signs with a *label*. An Amazon customer has an *account_ID*, a *main-credit-card*. 1 to 1 data can be represented in a single table.

1 to Many:

Like a band to its musicians, if each musician only belongs to one band. This requires two tables, a table of Bands & a table of Musicians. A field in the Musicians table could be the ID of the band to which they belong.

Many to Many

Bands in the 70's progressive rock genre (the pinnacle of human musical expression) swapped musicians all the time. A band had several musicians AND each musician often belonged to more than one band. To model this the *relational data base* way requires two tables of type 1 and a third table of type 2 to relate the two. This is the relationship already seen in the tables above.

SQL Command Cheatsheet

CREATE statement

```
CREATE TABLE bands (id INTEGER, name VARCHAR(64), label VARCHAR(64),
                    founding_city VARCHAR, created_at DATEIME,
updated_at DATETIME);

CREATE TABLE musicians (id INTEGER, label VARCHAR(64), first_name
VARCHAR,
                    last_name VARCHAR, main_instrument VARCHAR(64),
                    created_at DATEIME,updated_at DATETIME);

CREATE TABLE band-musician (id INTEGER, band_id INTEGER, musician_id
INTEGER,
                    created_at DATEIME, updated_at DATETIME );
```

If a musician can only belong to one band at a time, then musicians table above could have a "band_id" field. It would store the id of the musician's band. A third table wouldn't be needed.

.schema

```
Type .schema to the SQL command line find out what's in the database
```

SELECT statement

```
SELECT CustomerName,City FROM Customers;
```

SELECT DISTINCT Statement

```
SELECT DISTINCT City,Country FROM Customers;
```

City	Country
London	UK

Berlin	Germany

AND & OR Operators with WHERE

```
SELECT * FROM Customers
WHERE Country='Germany'
AND City='Berlin';
```

SELECT with WHERE and LIKE

```
SELECT * FROM Customers
WHERE Country LIKE '%United%'
```

SELECT and ORDER BY Keyword

```
SELECT column_name, column_name
FROM table_name
ORDER BY column_name ASC|DESC, column_name ASC|DESC;
```

SQL GROUP BY and COUNT

Give me a count of the number of tracks by unit price

```
SELECT unit_price,count(*) FROM tracks GROUP BY unit_price
```

INSERT INTO Statement

```
INSERT INTO Customers
    (CustomerName, ContactName, Address, City, PostalCode, Country)
```

```
VALUES
    ('Cardinal','Tom B. Erichsen','Skagen
21','Stavanger','4006','Norway');
```

Partial INSERT INTO Statement

```
INSERT INTO "dogs"
    ("license", "name", ...)
VALUES
    (?, ?, ...)
    [["license", "OH-9084736"], ["name", "Taj"], ...]
```

SQL UPDATE Statement

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

Use WHERE to specify which rows to update.

SQL DELETE Statement

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria
Anders';
```

SQL INNER JOIN

```
SELECT regions.region_name, states.state_name
   ...> FROM regions
   ...> INNER JOIN states
   ...> on regions.id=states.region_id
   ...> ORDER BY regions.id ASC;
```

Use WHERE to specify which rows to update.

COMPLEX SQL INNER JOIN

```
SELECT customers.first_name, customers.last_name, invoices.total
   ...> FROM customers
   ...> INNER JOIN invoices
   ...> on customers.id=invoices.customer_id
   ...> ORDER BY invoices.total DESC
   ...> LIMIT 1;
```

Given a table of customers and a table of invoices, return the customer (and invoice) with the highest invoice total.

Double JOIN

```
SELECT publishers.name
   ...> FROM publishers
   ...> JOIN books
   ...> on publishers.id=books.publisher_id
   ...>     JOIN authors
   ...>     on books.author_id=authors.id
   ...> WHERE authors.name='Robert Heinlein';
```

*Given publishers, books, and authors tables, return the publishers of all books written by Robert Heinlein.

SQL Injection

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

The example above, creates a select statement by adding a variable (txtUserId) to a select string. The variable is fetched from the user input (Request) to the page.

SQL Data Types

Data type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n)	
CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n)	
BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision
REAL	Approximate numerical, mantissa precision 7

FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time, depending on the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data