

Databases on the Web

- Many web applications, when they get complex enough, end up needing a database.
- Lots of ways to do this.

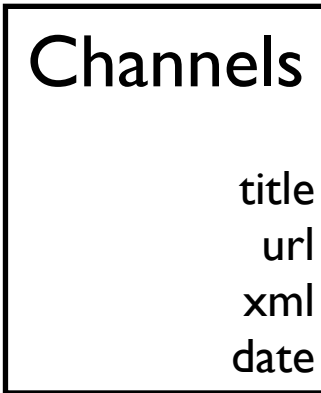
Types of Databases

- There's plain old files (open, print, close)
 - Organize EVERYTHING yourself.
- Common simple formats like
 - tab-delimited values
 - comma separated values
- Downsides?

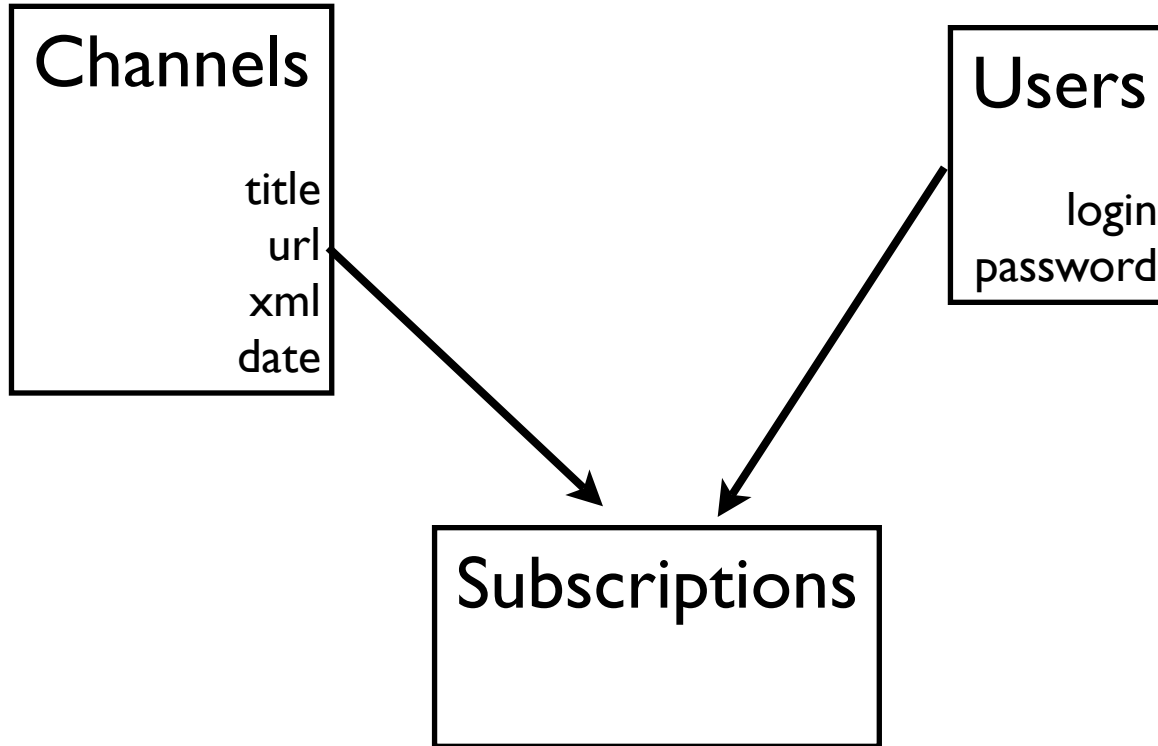
Types of Databases

- Simple databases like DBM, BerkeleyDB
- Quick lookup of key-value pairs
- Easy-to-use interface
- Very fast
- Scales up to very large amounts of data (gigabytes)
- Why would we need a relational database?

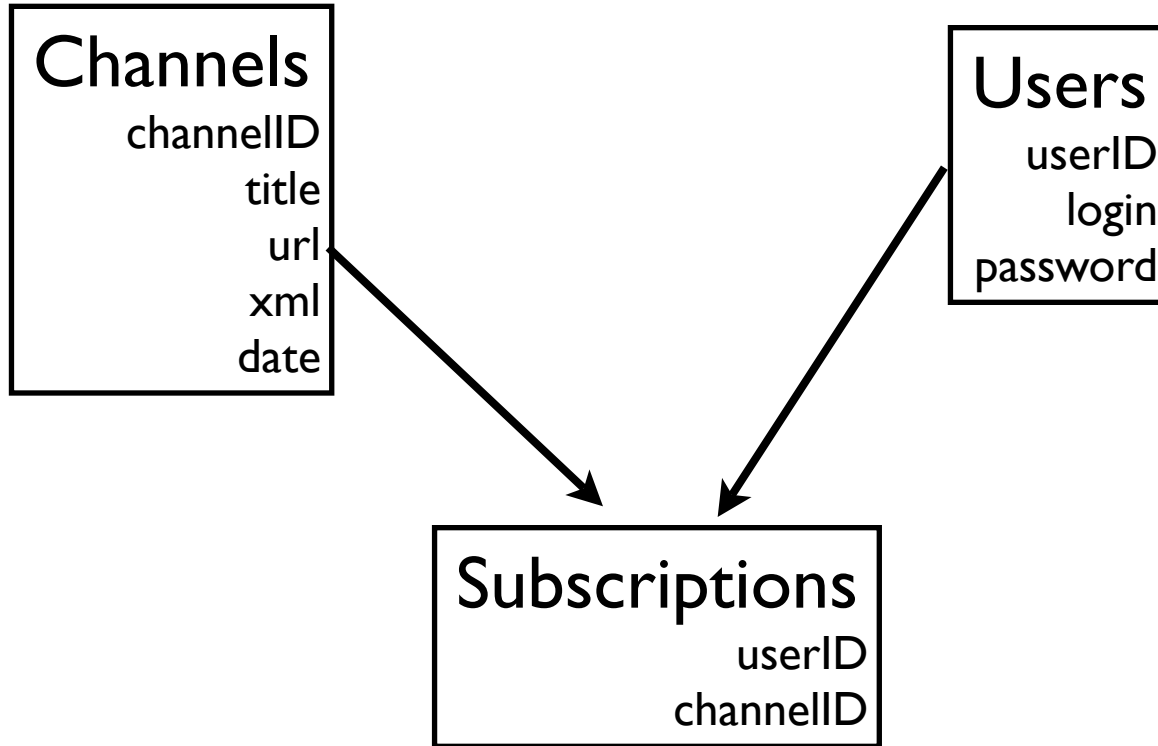
Relational Use Case



Relational Use Case



Relational Use Case



Relational Databases

- SQL
- Oracle, Informix, Sybase, Ingres, Postgres, mSQL, MySQL Illustra, DB2, Altera, ...
- Portable DB apps?
 - SQL spec incomplete
 - No standard front-ends or APIs
 - sybperl, oraperl, ingperl, ...
 - Could have app work with only 1 DB

Choosing a database

- We want scalability, modularity, and vendor-neutrality.
- Components: the browser, the web server, the database.
- Connect to diverse databases from a single application.
- Replacing a component should have small impact on the others.

Scenarios

- Capacity of Access is exceeded so switch to Oracle
- Connect to a Sybase-based legacy application
- Netscape lacks some desired feature so replace it with Apache

The Payoff

- If you write code that uses no RDBMS-specific calls...
- If you use only generic SQL...
- Then replacing one db with another is easy.
- You won't have to rewrite your program when your boss/customer/CIO makes you switch databases.

SQL92

- Anatomy of SELECT

```
SELECT  
FROM  
WHERE
```

SQL92

- Anatomy of SELECT

```
SELECT fieldnames (or *)  
FROM tablename  
WHERE conditions (key=value)
```

SQL92

- Get all users and passwords

```
SELECT login, password  
FROM users
```

SQL92

- Retrieve a password from the database

```
SELECT password
FROM users
WHERE login='dtreder'
```

```
# tip : quotes are not portable!
```

```
# mysql accepts either, but oracle does not!
```

```
# use DBI's quote feature to handle all quoting.
```

Performance w/ Perl

- Spawning a child process for each database operation.
- Establishing a new TCP/IP connection every time (perhaps even doing a nameserver lookup).
- Searching for and loading the DBI and DBD modules every time.
- In MS environments, ODBC may be an issue.

SQL92

- Anatomy of INSERT

```
INSERT INTO tablename  
SET fields  
VALUES ( values )
```

SQL92

- Anatomy of INSERT

```
INSERT INTO users
SET (login, password)
VALUES ( 'dtreder',
        'b7e055c6165da55c3e12c49ae5207455')
```

```
# Primary keys are not always portable!
# MySQL has a primary key autoincrement feature
# Oracle has sequences for a similar feature
# No one really uses natural primary keys
```

SQL92

- Anatomy of UPDATE

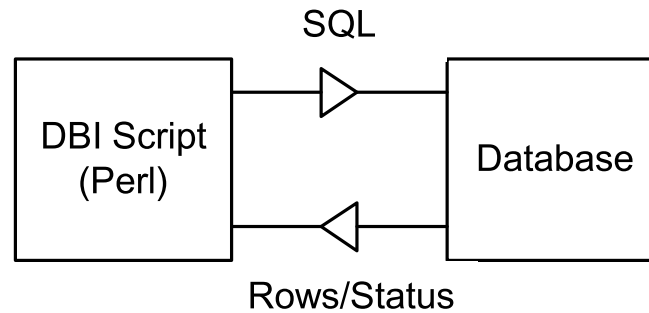
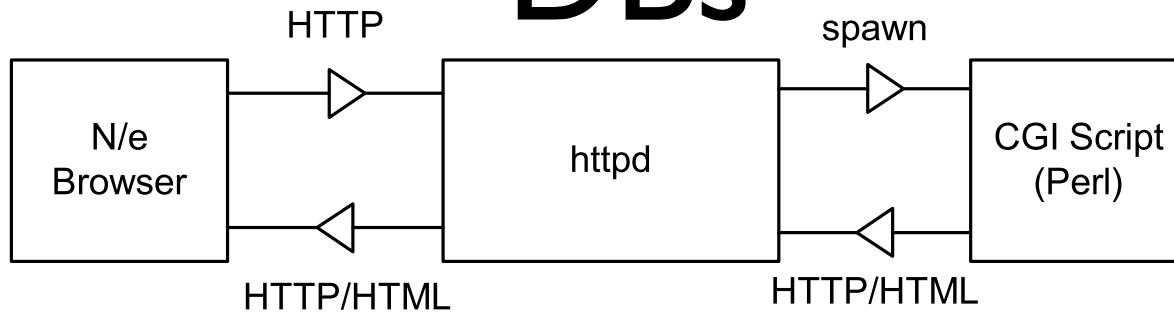
```
UPDATE tablename  
SET field=value, ...  
WHERE conditions
```

SQL92

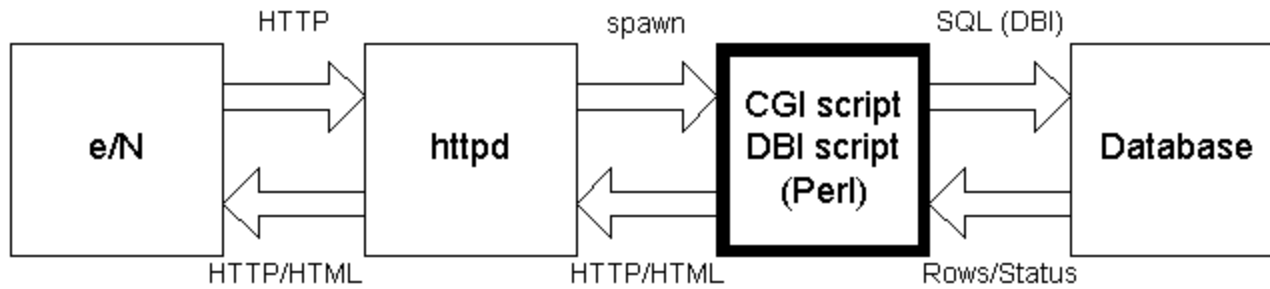
- Anatomy of UPDATE

```
UPDATE channels  
SET xml='<xml>blah blah</xml>'  
WHERE channelID=8;
```

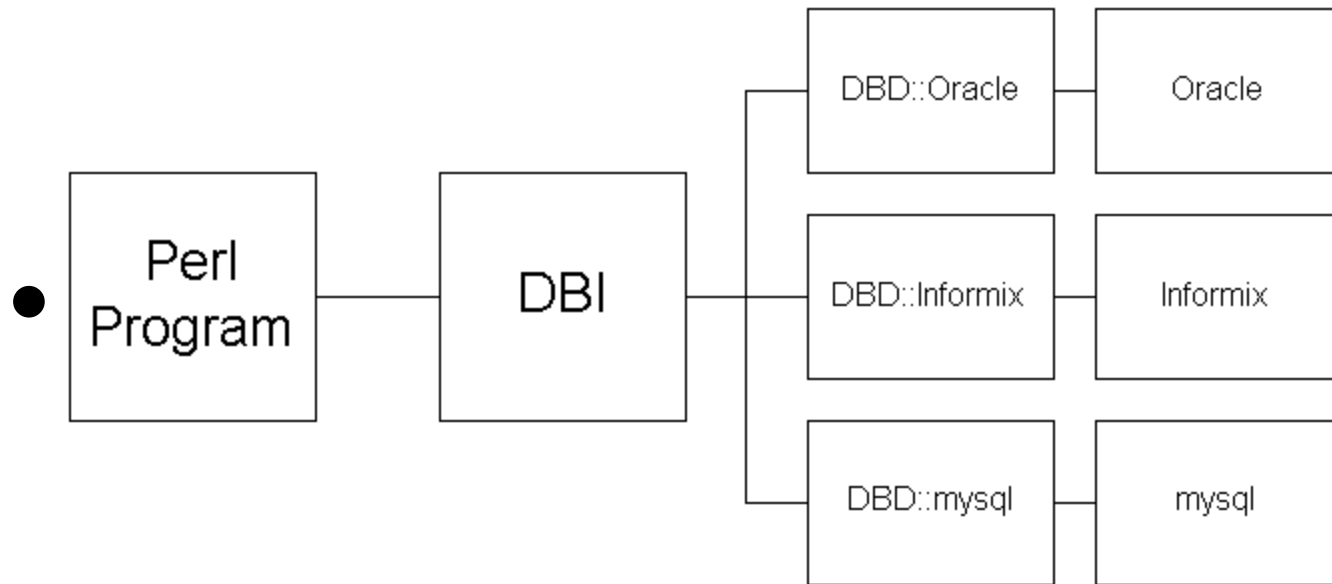
Perl as CGI language for DBs



Data Flows



DBI, DBD



DBI

- Written and maintained by Tim Bunce
- Provides a single, common API that encapsulates each vendor's specific API calls
- Example: The DBI call connect calls:
 - Oracle's ora_login
 - MySQL's mysql_connect
 - Appropriate for any other database

DBI

- DBI is an object class
- <http://www.symbolstone.org/technology/perl/DBI/index.html>
- Database specific functionality is in DBD modules, e.g. DBD::mysql

DBI Resources

- <http://www.symbolstone.org/technology/perl/DBI/doc/tpj5/index.html>
- <http://www.symbolstone.org/technology/perl/DBI/index.html>
- `perldoc DBI`

DBI->connect

- Establishing a database handle

```
use strict;
use DBI;
my $dbh = DBI->connect($dsn,$login_name, $password);

# getting the $dsn right is the only hard part
# For mysql, it's
# dbi:mysql:database_name
```

Pulling rows

```
my $dbh = DBI->connect($dsn,$login_name, $password);

$sql = "select login from users where userID=1";
$stmt = $dbh->prepare($sql);
$stmt->execute;
while(@row = $stmt->fetchrow_array) {

    # process fields in @row

}

$stmt->finish;
$dbh->disconnect;
```

Fetching rows

```
@row_ary = $sth->fetchrow_array;  
$ary_ref = $sth->fetchrow_arrayref;  
$hash_ref = $sth->fetchrow_hashref;  
  
$ary_ref = $sth->fetchall_arrayref;
```

binding values in SQL

- Most drivers support Placeholders and Bind Values.
- These drivers allow a database statement to contain placeholders, sometimes called parameter markers, that indicate values that will be supplied later, before the prepared statement is executed.
- This lets you reuse the SQL statement

More on binding

- Without using placeholders, the SQL statement would have to contain the literal values, and it would have to be re-prepared and re-executed for each row.
- With placeholders, the statement only needs to be prepared once. The bind values for each row can be given to the execute method each time it's called. By avoiding the need to re-prepare the statement for each row the application typically runs many times faster!

More on binding

- For example, an application might use the following to insert a row of data into the SALES table:

```
insert into sales (product_code, qty, price)
values (?, ?, ?)
```

```
#or the following, to select the description for a
#product:
```

```
select description
from products
where product_code = ?
```

Using bind parameters

```
my $sth = $dbh->prepare(
q{INSERT INTO sales (product_code, qty, price)
VALUES (?, ?, ?)}    ) || die $dbh->errstr;

while (<FILE>) {
    chomp;
    my ($product_code, $qty, $price) = split /,/;
    $sth->execute($product_code, $qty, $price)
        || die $dbh->errstr;
}
```