CS 580 Client-Server Programming
Spring Semester, 2009
Doc 11 SQL
4 March, 2009

Databases & Your server

You will be creating your own tables in your database for the server

Some students run databases on their own machines for development

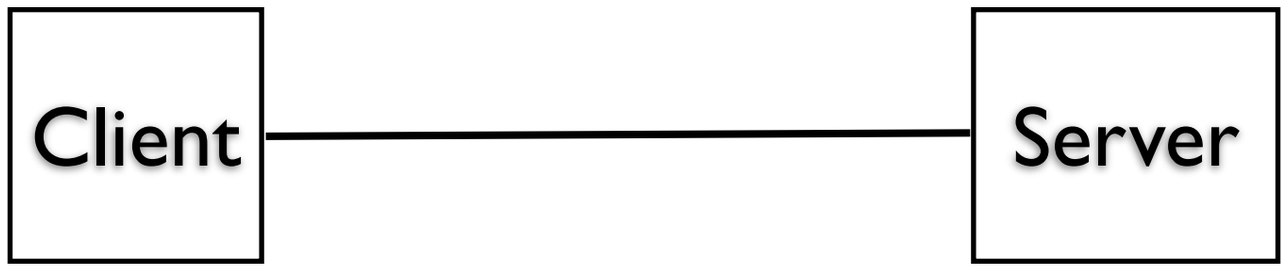
I will create PostgreSQL databases for this class

While you are not required to use that database the database you use must be accessible by your server when it is graded

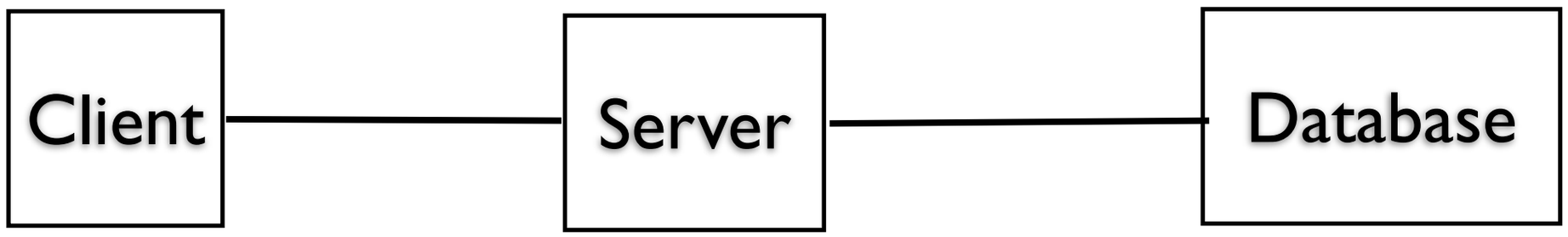
CS 514 in 59 slides

Jargon

2-Tier



3-Tier



More Jargon

Sometimes database means a program for managing data

Oracle Corporation is a database company.
MS Access is database.

Sometimes database means a collection of data

I keep a database of my CD collection on 3 by 5 cards

Sometimes database means a set of tables, indexes, and views

My program needs to connect to the Airline Reservation database, which uses Oracle

Some Reasons for Using a Database

Persistence of data

Sharing of data between programs

Handle concurrent requests for data access

Transactions that can be rolled back

Report generation

Types of Databases

Relational

Data is stored in tables

Object-Oriented

Tables can be subclassed

Programmer can define methods on tables

Object

Objects are stored in the database

Relational, Object-Oriented Databases and SQL

Database consists of a number of tables

Table is a collection of records

Each Column of data has a type

firstname	lastname	phone	code
John	Smith	555-9876	2000
Ben	Oker	555-1212	9500
Mary	Jones	555-3412	9900

Use Structured query language (SQL) to access data

Some Available Databases

Oracle
DB2
SQL Server
Access
Informix
Ingres
InterBase
Sybase
FileMaker Pro
FoxPro
Paradox
dBase

Open Source Databases
MySQL
PostgresSQL

SQL History

Dr. E. F. Codd develops relational database model

Early 1970's

IBM System R relational database

Mid 1970's

Contained the original SQL language

First commercial database - Oracle 1979

SQL was aimed at:

Accountants

Business people

SQL92

First commonly followed standard

ANSI X3.135-1992

SQL2

ISO/IEC 9075-1 through 5

New SQL standard

MySQL & PostgreSQL

Open source databases

http://www.mysql.com/

http://www.postgresql.org/

Above site have free downloads and documentation

Historically I used MySql for this course. While it had some defects MySql was easier to install, easier to use on simple situations, had better documentation, and was far more common than PostgreSql. The situation has changed a lot. PostgreSql is now as easy to install, its documentation is as good as MySql, it runs faster than MySql, and has better support for transactions. There also is the issue of Sun's purchase of MySql and some lost of direction for the free version of MySql. So at least personally I have stopped using MySql and don't provided MySql accounts for students.

MySQL – Connecting to the Database

Can be done with:

- Mysql command line tool - mysql
- GUI clients
- Program

GUI Clients

If done well are very useful

There are many of these

I use DbVisualizer, & CocoaMySQL

DbVisualizer if Java based so runs on may platforms

http://www.dbvis.com/products/dbvis/

SQL Syntax

Names

Databases, tables columns & indexes have names

Legal Characters

Alphanumeric characters, '_', '$'

Names can start with:

- Letter
- Underscore
- Letter with diacritical marks and some non-latin letters

Name length

63 characters – default in PostgreSQL
64 characters - MySQL

Names are not case sensitive

Data Types

Numeric Values

Integer - decimal or hex

Floating-point - scientific & 12.1234

String Values

‘this is a string’ PostgreSQL

‘this is a string’ “this is also a string" MySQL

Sequence	Meaning
\'	Single quote
\b	Backspace
\n	Newline
\r	Tab
\\	Backslash
\xxxx	Character were xxxx is the octal of ASCII code (PostgreSQL)

Including a quote character in a string

Double quote the character

'Don''t do it'

Escape the quote character with a backslash

'Don\'t do it'

Comments

```
-- this is a comment in MySQL and PostgreSQL

/* this is also a comment in MySQL and PostgreSQL */

# this is a comment in MySQL
```

Numeric Data Types

Type name	Description	Range
smallint	Fixed-precision	-32768 to +32767
integer	Usual choice for fixed-precision	-2147483648 to +2147483647
bigint	Very large range fixed-precision	-9223372036854775808 to 9223372036854775807
decimal	user-specified precision, exact	no limit
numeric	user-specified precision, exact	no limit
real	variable-precision, inexact	6 decimal digits precision
double precision	variable-precision, inexact	15 decimal digits precision
serial	autoincrementing integer	1 to 2147483647

Numeric(10, 2) defines a number with maximum of 10 digits with 2 of the 10 to the right of the decimal point

12345678.91

decimal and numeric are different names for the same type

String Types

Type	Description
char(n)	Fixed-length blank padded
varchar(n)	Variable-length with limit
text	Variable unlimited length
bytea (PostgreSQL)	Variable (not specifically limited) length binary string
blob (MySQL)	Variable (not specifically limited) length binary string

CHAR & VARCHAR are the most common string types

CHAR is fixed-width

Shorter strings are padded

TEXT can be any size

PostgreSQL limits a string to 1GB in storage space

MySQL limits CHAR and VARCHAR to 255 characters

Date & Time Types - PostgreSQL

Type	Description
timestamp [(p)] without time zone	both date and time
timestamp [(p)] [with time zone]	both date and time
interval [(p)]	for time intervals
date	dates only
time [(p)] [without time zone]	times of day only
time [(p)] with time zone	times of day only

(p) indicates optional number of fractional digits retained in the seconds field

Date Formats - PostgreSQL

Example	Description
January 8, 1999	Unambiguous
1999-01-08	ISO-8601 format, preferred
1/8/1999	U.S.; read as August 1 in European mode
8/1/1999	European; read as August 1 in U.S. mode
1/18/1999	U.S.; read as January 18 in any mode
19990108	ISO-8601 year, month, day
990108	ISO-8601 year, month, day
1999.008	Year and day of year
99008	Year and day of year
J2451187	Julian day
January 8, 99 BC	Year 99 before the Common Era

Setting Date Formats - PostgreSQL

SET DateStyle TO ‘US’
SET DateStyle TO ‘NonEuropean’

Sets date format to month day year

SET DateStyle TO ‘European’

Sets date format to day month year

Default is ISO style

Dates – MySQL

DATETIME – ‘YYYY-MM-DD HH:MM:SS’ format

DATE – ‘YYYY-MM-DD’ format

TIMESTAMP

- Changed in MySQL 4.1
- Basically now is same as DATETIME

Common SQL Statements

SELECT	Retrieves data from table(s)
INSERT	Adds row(s) to a table
UPDATE	Changes field(s) in record(s)
DELETE	Removes row(s) from a table Data Definition
CREATE TABLE	Define a table and its columns(fields)
DROP TABLE	Deletes a table
ALTER TABLE	Adds a new column, add/drop primary key
CREATE INDEX	Create an index
DROP INDEX	Deletes an index
CREATE VIEW	Define a logical table from other table(s)/view(s)
DROP VIEW	Deletes a view

SQL is not case sensitive

Examples That Follow

Will use mysql command line tool

Used the command

mysql -h host -u user -p

to conntect to the database, where host and user are given the correct value

On rohan the full name of command is:

/opt/local/mysql/bin/mysql

Some examples will also show postgresSQL text client

CREATE DATABASE

General Form

CREATE DATABASE [IF NOT EXISTS] db_name
[create_specification [, create_specification] ...]

create_specification:
[DEFAULT] CHARACTER SET charset_name
| [DEFAULT] COLLATE collation_name

Example

mysql> create database lectureExamples;
Query OK, 1 row affected (0.00 sec)

PosgreSQL Example

```
Al 15->psql -h bismarck.sdsu.edu cs580whitney cs580whitney
Password:
Welcome to psql 7.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit


cs580whitney=> create database lectureExamples;
ERROR:  permission denied to create database
cs580whitney=>
```

Student accounts do not have authority to create new databases

USE

Sets a default database for subsequent queries

General Form

USE db_name

Example

```
mysql> use lectureExamples;
Database changed
```

CREATE table

General Form

```
CREATE TABLE table_name (
    col_name  col_type [ NOT NULL | PRIMARY
KEY]
    [, col_name col_type
          [ NOT NULL | PRIMARY KEY]]*
)
```

Example

```
mysql> CREATE TABLE students
       (
       firstname  CHAR(20) NOT NULL,
       lastname   CHAR(20),
       phone      CHAR(10),
       code       INTEGER
       );
```

```
mysql> CREATE TABLE codes
       (
       code       INTEGER,
       name       CHAR(20)
       );
```

PostgreSQL Example

```
cs580whitney=> CREATE TABLE students
cs580whitney-> (
cs580whitney(> firstname  CHAR(20) NOT NULL,
cs580whitney(> lastname   CHAR(20),
cs580whitney(> phone      CHAR(10),
cs580whitney(> code       INTEGER
cs580whitney(> );

CREATE TABLE

cs580whitney=> select * from students;
 firstname | lastname | phone | code
-----------+----------+-------+------
(0 rows)
```

Select

Gets data from one or more tables

General Form

```
SELECT [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
    [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE]
    [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
    [DISTINCT | DISTINCTROW | ALL]
  select_expression,...
  [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
  [FROM table_references
    [WHERE where_definition]
    [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC], ...
      [WITH ROLLUP]]
    [HAVING where_definition]
    [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,...]
    [LIMIT [offset,] row_count | row_count OFFSET offset]
    [PROCEDURE procedure_name(argument_list)]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

Select Example

```
mysql> SELECT * FROM students;
Empty set (0.00 sec)
```

Insert

Add data to a table

General Form

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES ((expression | DEFAULT),...),(...),...
    [ ON DUPLICATE KEY UPDATE col_name=expression, ... ]
```

Insert Examples

```
mysql> INSERT
    INTO students (firstname, lastname, phone, code)
    VALUES ('Roger', 'Whitney', '594-3535', 2000 );

mysql> INSERT
    INTO codes (code, name)
    VALUES (2000, 'marginal' );

mysql> SELECT * FROM students;
+----------+---------+---------+------+
| firstname | lastname | phone    | code |
+----------+---------+---------+------+
| Roger     | Whitney  | 594-3535 | 2000 |
+----------+---------+---------+------+
1 row in set (0.01 sec)
```

More Select Examples

```
mysql> SELECT firstname , phone FROM students;
+-----------+----------+
| firstname | phone    |
+-----------+----------+
| Roger     | 594-3535 |
+-----------+----------+
1 row in set (0.00 sec)

mysql> SELECT lastname, name
       FROM students, codes
       WHERE students.code = codes.code;

+----------+----------+
| lastname | name     |
+----------+----------+
| Whitney  | marginal |
+----------+----------+
1 row in set (0.00 sec)
```

More Select Examples

```
mysql> SELECT students.lastname, codes.name
       FROM students, codes
       WHERE students.code = codes.code;
```

```
+----------+----------+
| lastname | name     |
+----------+----------+
| Whitney  | marginal |
+----------+----------+
1 row in set (0.00 sec)
```

Update

Modify existing data in a database

General Form

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name [, tbl_name ...]
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
```

Example

```
mysql> UPDATE students
       SET firstname='Sam'
       WHERE lastname='Whitney';

Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Few More SQL Commands

mysql> **ALTER TABLE students ADD column foo CHAR(40);**
Query OK, 1 row affected (0.03 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> **DROP TABLE students;**
Query OK, 0 rows affected (0.01 sec)

mysql> **DROP DATABASE lectureexamples;**
Query OK, 0 rows affected (0.00 sec)

An Example

PostgreSQL Version

```
CREATE TABLE faculty (
   name  CHAR(20) NOT NULL,
   faculty_id  SERIAL   PRIMARY KEY
);
```

name	faculty_id
Whitney	1
Beck	2
Anantha	3

MySQL Version

```
CREATE TABLE faculty (
   name  CHAR(20) NOT NULL,
   faculty_id  INTEGER AUTO_INCREMENT   PRIMARY KEY
);
```

Indices

Indices make accessing faster

Primary keys automatically have an index

The CREATE INDEX command creates indices

```
CREATE INDEX faculty_name_key on faculty (name);
```

Adding Values

```
INSERT INTO  faculty ( name) VALUES    ('Whitney');
INSERT INTO  faculty ( name) VALUES    ('Beck');
INSERT INTO  faculty ( name) VALUES    ('Anantha');
INSERT INTO faculty ( name) VALUES ('Vinge');

select * from faculty;
```

Result

```
          name           | faculty_id
-------------------------+------------
 Whitney                 |          1
 Beck                    |          2
 Anantha                 |          3
 Vinge                   |          4
(4 rows)
```

Second Table

start_time	end_time	day	faculty_id	office_hour_id
10:00	11:00	Wed	1	1
8:00	12:00	Mon	2	2
17:00	18:30	Tue	1	3
9:00	10:30	Tue	3	4
9:00	10:30	Thu	3	5
15:00	16:00	Fri	1	6

name	faculty_id
Whitney	1
Beck	2
Anantha	3
Vinge	4

Generating Second Table

PostgreSQL

```
CREATE TABLE office_hours (
    start_time    TIME NOT NULL,
    end_time    TIME NOT NULL,
    day    CHAR(3) NOT  NULL,
    faculty_id INTEGER REFERENCES faculty,
    office_hour_id    SERIAL    PRIMARY KEY
);
```

MySQL

```
CREATE TABLE office_hours (
    start_time    TIME NOT NULL,
    end_time    TIME NOT NULL,
    day    CHAR(3) NOT  NULL,
    faculty_id INTEGER REFERENCES faculty,
    office_hour_id INTEGER AUTO_INCREMENT PRIMARY KEY
);
```

Adding Office Hours

Simple Insert

```
INSERT
   INTO office_hours ( start_time, end_time, day, faculty_id )
   VALUES ( '10:00:00',  '11:00:00' ,  'Wed',  1 );
```

The problem is that we need to know the id for the faculty

Adding Office Hours

Using Select

```
INSERT INTO
    office_hours  (start_time, end_time, day, faculty_id )
SELECT
    '8:00:00' AS start_time,
    '12:00:00' AS end_time,
    'Mon' AS day,
    faculty_id AS faculty_id
FROM
    faculty
WHERE
    name = 'Beck'
```

Selecting Office Hours

```
SELECT
    name, start_time, end_time, day
FROM
    office_hours, faculty
WHERE
    faculty.faculty_id = office_hours.faculty_id;
```

name	start_time	end_time	day
Whitney	10:00:00	11:00:00	Wed
Beck	08:00:00	12:00:00	Mon
Whitney	17:00:00	18:30:00	Tue
Whitney	15:00:00	16:00:00	Fri
Anantha	09:00:00	10:30:00	Tue
Anantha	09:00:00	10:30:00	Thu

PostgreSQL only

```
SELECT
    name AS Instructor,
    TEXT(start_time) || ' to ' || TEXT(end_time) AS Time,
    day AS Day
FROM
    office_hours, faculty
WHERE
    faculty.faculty_id = office_hours.faculty_id
ORDER BY
    Name
```

Instructor	Time	Day
Anantha	09:00:00 to 10:30:00	Tue
Anantha	09:00:00 to 10:30:00	Thu
Beck	08:00:00 to 12:00:00	Mon
Whitney	10:00:00 to 11:00:00	Wed
Whitney	17:00:00 to 18:30:00	Tue
Whitney	15:00:00 to 16:00:00	Fri

Sample Selection

```
SELECT
    name, start_time, end_time, day
FROM
    office_hours, faculty
WHERE
    faculty.faculty_id = office_hours.faculty_id
 AND
    start_time > '09:00:00'
 AND
    end_time < '16:30:00'
ORDER BY
    Name;
```

name	start_time	end_time	day
Whitney	10:00:00	11:00:00	Wed
Whitney	15:00:00	16:00:00	Fri

Joins

People

id	first_name	last_name
1	Roger	Whitney
2	Leland	Beck
3	Carl	Eckberg

Email_Addresses

id	user_name	host	person_id
1	beck	cs.sdsu.edu	2
2	whitney	cs.sdsu.edu	1
3	whitney	rohan.sdsu.edu	1
4	foo	rohan.sdsu.edu	

Inner Join

Only uses entries linked in two tables

first_name	last_name	user_name	host
Leland	Beck	beck	cs.sdsu.edu
Roger	Whitney	whitney	cs.sdsu.edu
Roger	Whitney	whitney	rohan.sdsu.edu

```
select
        first_name, last_name, user_name, host
from
        people, email_addresses
where
        people.id = email_addresses.person_id;
```

```
select
        first_name, last_name, user_name, host
from
        people inner join email_addresses
on
        (people.id = email_addresses.person_id);
```

Outer Left Join

Use all entries from the left table

first_name	last_name	user_name	host
Leland	Beck	beck	cs.sdsu.edu
Roger	Whitney	whitney	cs.sdsu.edu
Roger	Whitney	whitney	rohan.sdsu.edu
Carl	Eckberg		

```
select
      first_name, last_name, user_name, host
from
      people left outer join email_addresses
on
      (people.id = email_addresses.person_id);
```

Right Outer Join

Use all entries from the right table

first_name	last_name	user_name	host
Leland	Beck	beck	cs.sdsu.edu
Roger	Whitney	whitney	cs.sdsu.edu
Roger	Whitney	whitney	rohan.sdsu.edu
		foo	rohan.sdsu.edu

```
select
      first_name, last_name, user_name, host
from
      people right outer join email_addresses
on
      (people.id = email_addresses.person_id);
```

A right outer join B == B left outer join A

The following two statements are equivalent

```
select
	first_name, last_name, user_name, host
from
	people right outer join email_addresses
on
	(people.id = email_addresses.person_id);
```

```
select
	first_name, last_name, user_name, host
from
	email_addresses left outer join people
on
	(people.id = email_addresses.person_id);
```

Normal forms

Defined by Dr. E. F. Codd in 1970

Reduce redundant data and inconsistencies

First Normal Form (1NF)

An entity is in the first normal form when all its attributes are single valued

Name	OfficeHour1	OfficeHour2	OfficeHour3
Whitney	10:00-11:00 W	17:00-18:30 Tu	15:00-16:00 Fri
Beck	8:00-12:00 M		
Anantha	9:00-10:30 Tu	9:00-10:30 Thu	

What if someone has more than 3 office hours?
Wasted space for those that have fewer office hours

Not is 1NF since office hours are repeated

In 1NF

Faculty

name	faculty_id
Whitney	1
Beck	2
Anantha	3

Office Hours

start_time	end_time	day	faculty_id	office_hour_id
10:00	11:00	Wed	1	1
8:00	12:00	Mon	2	2
17:00	18:30	Tue	1	3
9:00	10:30	Tue	3	4
9:00	10:30	Thu	3	5
15:00	16:00	Fri	1	6

Second Normal Form (2NF)

cd_title	artist	music_type	cd_id
Songs from the Trilogy	Glass	Modern Classical	1
I Stoten	Falu Spelmanslag	Swedish	2
Photographer	Glass	Modern Classical	3

An entity is in the second normal form if:

It is in 1NF and
All non-key attributes must be fully dependent on the entire primary key

Table is not in 2NF since different CDs

Can have the same artists
Can have same music type

Example 2

Name	Time	Days	Term	Schedule Number
CS635	1700-1815	MW	Spring01	9461
CS651	1700-1815	MW	Spring01	9472
CS672	1700-1815	MW	Spring01	9483
CS683	1830-1945	MW	Spring01	9494
CS696	1530-1645	MW	Spring01	9505
CS696	1830-1945	MW	Spring01	9516
CS696	1530-1645	TTh	Spring01	9520

At SDSU the schedule number uniquely identifies a course in a semester
So the term and schedule number uniquely identifies a course at SDSU
We can use term and schedule as the primary key

The table is in 1NF but not 2NF

Name, Time and Days are not fully dependent on the primary key

Schedule in 2NF

Schedule

course_id	time_id	term_id	schedule_number
1	1	2	9461
2	1	2	9472
3	1	2	9483
4	2	2	9494

Term

semester	year	term_id
Fall	2000	1
Spring	2001	2

Courses

course	title	name_id
CS635	Adv Obj Orient Dsgn Prog	1
CS651	Adv Multimedia Systems	2
CS683	Emerging Technologies	3
CS696	Writing Device Drivers	4

Time

start_time	end_time	days	time_id
17:00:00	18:15:00	MW	1
18:30:00	19:45:00	MW	2
15:30:00	16:45:00	MW	3
15:30:00	16:45:00	TTh	4
Etc.			

Comments about Previous Slide

The schedule table is now in 2NF

What about the other tables?

If not how would you fix them?

Can you find a better way to decompose the original table?

Third Normal Form (3NF)

Customer

Name	Address	City	State Name	State abbreviation	zip	id

An entity is in third normal form if

It is in 2NF and
All non-key attributes must only be dependent on the primary key

State abbreviation depends on State Name

Table is not in 3NF