CSC 546: Client/Server Fundamentals

Fall 2000

Client/server databases & SQL

- relational databases
- Standard Query Language (SQL)
  - tables, updates, queries, transactions
- object databases
- SQL API's
  - embedded SQL, module languages, call level interfaces

Client/server databases

in many real-world applications, the server maintains a shared database

- clients access the shared data, make updates

- many vendors provide networked database systems
  underlying most is the Standard Query Language (SQL)

SQL is a standard for specifying relational database operations

- it is NOT a communications protocol

- database software must convert SQL commands into the underlying RPC invocations or socket commands or …

SQL is popular because:

1. it is an industry standard, AND
2. it allows the user to manipulate a database without worrying about its organization

Relational databases

virtually all databases are based on the relational model

- data is organized into *tuples* of (name, value) pairs

```
{ (name, "smith"), (password, "foobar"), (average, 87.5) }
```

- *relations* are collections of tuples having the same schema
- SQL uses *tables* to encode relations

name	password	average
'smith'	'foobar'	87.5
'jones'	'abc123'	59.7
'vneumann'	'x?1@#zz'	99.8
'turing'	'????'	75.0

SQL

SQL is a declarative language in which the user (or an application) can specify what data they want

- it does not specify how the data is to be stored, organized, or retrieved
- using SQL commands, a user can access/manipulate data in any database engine that supports SQL

CREATE: used to create a table

```
CREATE TABLE Grades
    (name     VARCHAR,
     password VARCHAR,
     average  FLOAT);
```

this SQL command would create a table named Grades

- each entry (relation) in the table has three tuples with specific types
- VARCHAR specifies a variable length sequence of chars (i.e., a string)

SQL data types

- `CHARACTER` for a single character
- `CHAR(N)` for a string of n chars (padded with spaces)
- `VARCHAR` for variable length strings (up to 255 chars)

- `DECIMAL` for exact numbers, 16 significant digits
- `SMALLINT` for exact numbers, abs value < 2^{15}
- `INTEGER` for exact numbers, abs value < 2^{31}

- `FLOAT` for approx. numbers, 10^{-38} < abs value < 10^{38}

- `DATE` for date, nonzero value < `9999-13-32`
- `TIME` for time, pos. value < `24.60.62.000000`
- `TIMESTAMP` for time stamp, value < `9999.13.32.24.60.62.000000`

.
.
.

Altering a table

ALTER: used to change the schema for existing tables

```
ALTER Grades
    ADD grade CHAR(2),
    DROP password;
```

managing the evolution of schema as alterations are made is tricky:

- existing data must be reformatted
- data in other tables may depend upon the current schema
- altering the table requires taking it offline temporarily

name	average	grade
'smith'	87.5	NULL
'jones'	59.7	NULL
'vneumann'	99.8	NULL
'turing'	75.0	NULL

Updating a table

UPDATE: used to change the values in existing rows of the table

```
UPDATE Grades
    SET grade = 'F'
    WHERE name = 'smith';

UPDATE Grades
    SET grade = 'A'
    WHERE average >= 90.0;

UPDATE Grades
    SET grade = 'C+'
    WHERE grade = NULL;
```

name	average	grade
'smith'	87.5	'F'
'jones'	59.7	'C+'
'vneumann'	99.8	'A'
'turing'	75.0	'C+'

Deleting table entries

DELETE: used to delete a row of the table

```
DELETE FROM Grades
    WHERE grade = 'F';
```

name	average	grade
'jones'	59.7	'C+'
'vneumann'	99.8	'A'
'turing'	75.0	'C+'

Inserting table entries

INSERT: used to add a row to the table

```
INSERT INTO Grades
    VALUES ('hopper', 83.4, 'B');

INSERT INTO Grades
    (name, grade)
    VALUES ('gates', 'D');
```

name	average	grade
'jones'	59.7	'C+'
'vneumann'	99.8	'A'
'turing'	75.0	'C+'
'hopper'	83.4	'B'
'gates'	NULL	'D'

Querying a table

SELECT: used to selectively access the contents of the table

```
SELECT name, average
    FROM Grades;
```

name	average
'jones'	59.7
'vneumann'	99.8
'turing'	75.0
'hopper'	83.4
'gates'	NULL

```
SELECT name, average
    FROM Grades
    WHERE grade = 'C+';
```

name	average
'jones'	59.7
'turing'	75.0

```
SELECT name
    FROM Grades
    WHERE grade = 'A'
    OR grade = 'B';
```

name
'vneumann'
'hopper'

Querying a table (cont.)

can sort the results of a select

```
SELECT name, grade
    FROM Grades
    ORDER BY grade;
```

name	grade
'vneumann'	'A'
'hopper'	'B'
'jones'	'C+'
'turing'	'C+'
'gates'	'D'

```
SELECT name, average
    FROM Grades
    ORDER BY average DESC;
```

name	average
'vneumann'	99.8
'hopper'	83.4
'turing'	75.0
'jones'	59.7
'gates'	NULL

Querying a table (cont.)

can use select to *equi-join* a multi-table query

name	average	grade
'jones'	53.7	'F'
'turing'	75.0	'C+'
'hopper'	83.4	'B'
'gates'	48.2	'F'

Grades

name	address	phone
'jones'	'????'	'280-1234'
'vneumann'	'????'	'280-0000'
'turing'	'????'	'555-9999'
'hopper'	'????'	'292-0101'
'gates'	'????'	'280-6666'

Contacts

```
SELECT Grades.name, Contacts.phone
    FROM Grades, Contacts
    WHERE Grades.name = Contacts.name
    AND Grades.grade = 'F';
```

name	phone
'jones'	'280-1234'
'gates'	'280-6666'

Transaction processing

in a client/server system, server may service requests concurrently

a *transaction* is a sequence of operations that must be performed as if they were a single operation

- atomicity: all ops must complete successfully or none
  must be able to undo operations if failure
- consistency: applying the ops must leave the database in a consistent state
  integrity constraints must be checked after each transaction
- isolation: concurrent transactions must not affect each other until completed
  usually implemented using locks on entries, danger of deadlock
- durability: once completed, transaction updates must be permanent
  must write to disk after each transaction

COMMIT: causes the database to confirm that all prior transactions are saved

ROLLBACK: causes the database to undo all transactions since last COMMIT

SQL versions

SQL-86

- original SQL commands and syntax

SQL-87

- referential integrity constraints between tables
- referential constraints across rows of a table
- default column values for new rows

SQL-89

- data definition language (DDL):
  commands for defining tables, views, indexes, integrity constraints
- data manipulation language (DML)
  commands for selecting, updating, deleting, inserting rows of data
- data control language (DCL)
  commands for managing recovery, locking, security, and transactions
- embedded SQL

SQL versions (cont.)

SQL-92

- SQL network connection establishment
- more granular transaction and locking controls
- dynamic SQL
- new datatypes (blobs, varchar, date, time, timestamp)
- standardized database catalogs and temporary tables
- multiple types of joins (outer, inner, union, cross)
- standardized error codes
- scrollable cursors

SQL-3

- callable level interface (CLI)
- language bindings
- major object-oriented SQL extensions (MOOSE)
- multimedia SQL
- user-defined data types
- persistent and sensitive cursors
- temporary views
- user-defined security roles
- subtables and supertables

Object databases

object-oriented databases extend the relational model with more generalized object concepts

- major object-oriented SQL extensions (MOOSE)
- object database standard (ODMG-93)

ODMG-93 specifies:

- Object Definition Language (ODL)
  defines interfaces to object types within an OODB (extends CORBA IDL)
- Object Query Language (OQL)
  provides declarative access to objects
- Object Manipulation Language (OML)
  for retrieving objects from the database and modifying them

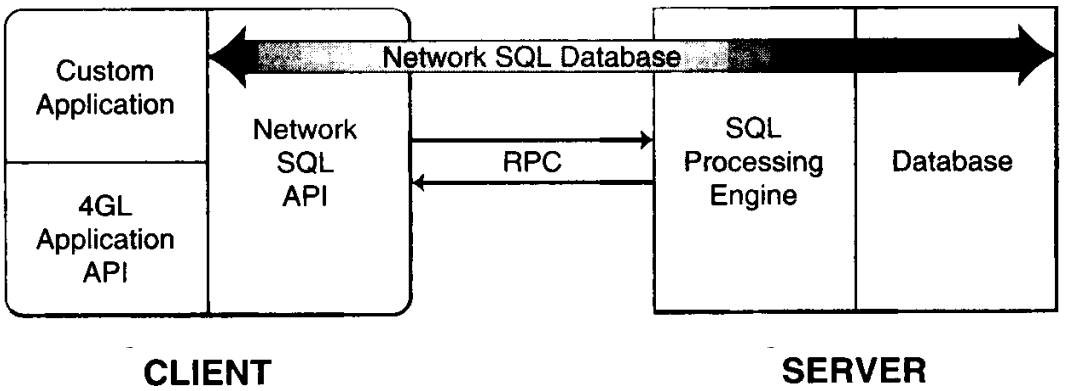
```
interface Car: Vehicle
( extent cars
  keys license_plate): persistent
{ attribute String make;
  attribute String license_plate;
  attribute Integer fuel_level;
  relationship Owner owned_by inverse Owner::owns;
  void add_fuel(in integer amount) raises (tank_full);
  . . .
}
```

```
select warn_list (o.name, c.license_plate)
    from c in Car,
         o in Owner,
         x in c.owned_by
    where c.fuel_level = 0;
```

SQL API's

accessing a database over a network (without resorting to RPCs)

- an Application Programming Interface (API) defines the client's language
- the application (client) packages SQL ops and invokes them using the API
- off-the-shelf SQL API routines perform the actual communication with the database server, return the results



issues when accessing a database from a programming environment:

- most programming languages are procedural, SQL is declarative
- differences in command syntax
  handled via language bindings, e.g., bind SQL data to C++ data type
- differences in how data is accessed (e.g., sequential vs. concurrent access)
  handled by providing cursors, pointer into the data for program
- differences in return codes and error indications

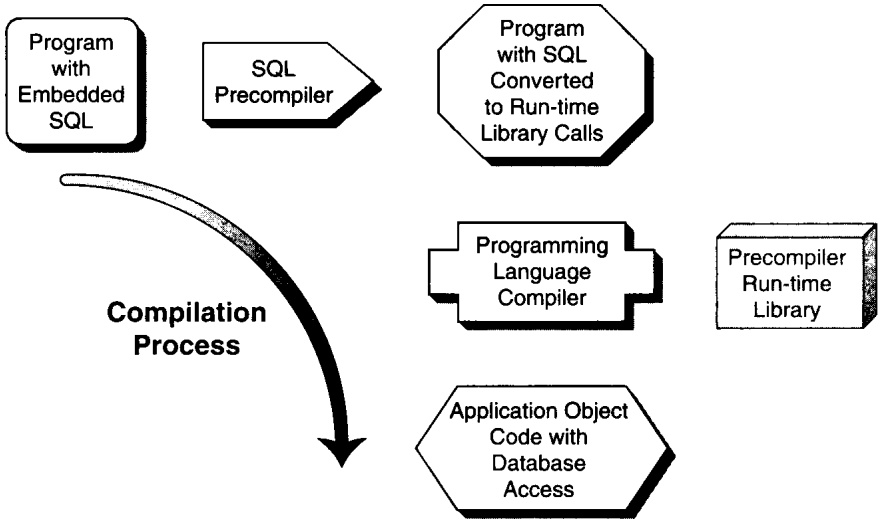
Embedded SQL

the most common type of SQL API is *embedded SQL*

- programmer embeds SQL commands in program (using C++, Java, perl, …)

```
// C++ code
EXEC SQL SELECT average FROM Grades WHERE name = 'smith';
// more C++ code
```

- a preprocessor, provided by database vendor, compiles the embedded SQL commands into database library calls
- once preprocessed, the program can be compiled and executed



Embedded SQL (cont.)

- language bindings specify the mappings between SQL data types and the data types of the programming language

- note: an SQL SELECT can return an entire rows & columns of data
  - in a procedural language, need the ability to traverse systematically
  - a *cursor* is a pointer into the returned data, can be used to traverse

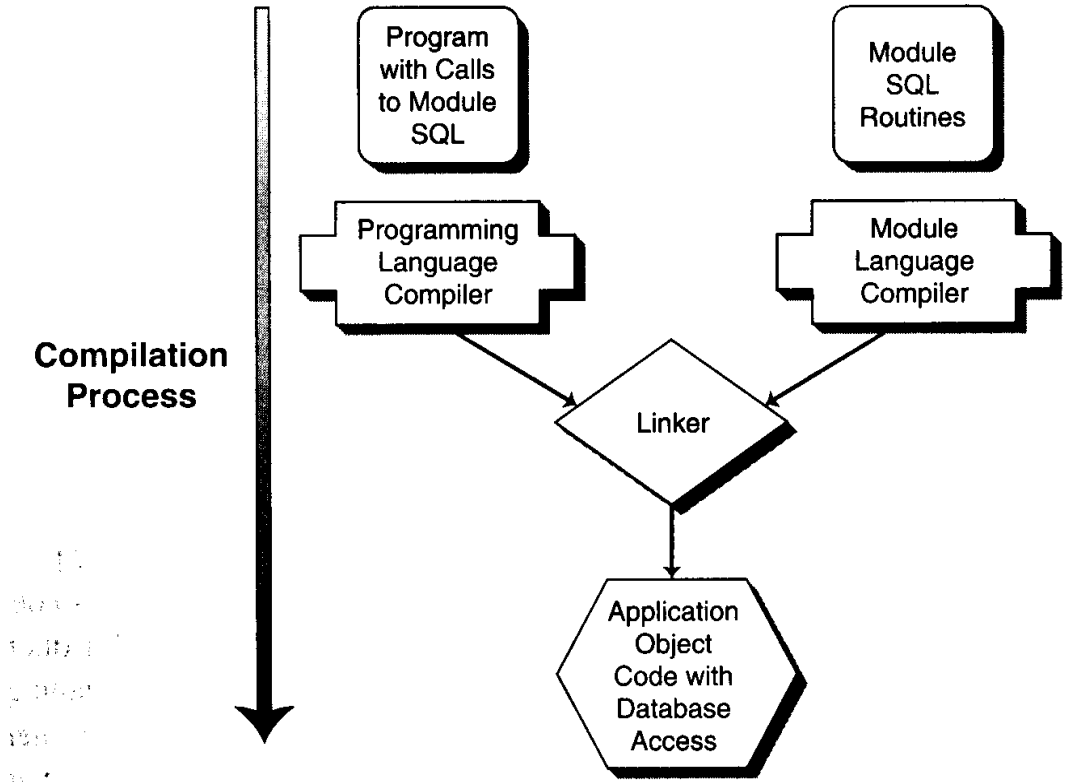
```
EXEC SQL BEGIN DECLARE SECTION;
  string name;
  double average;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE stu CURSOR FOR
  SELECT name, average FROM Grades WHERE class = 'csc546';

EXEC SQL OPEN stu;
for (int i = 0; i < numStudents; i++) {
    EXEC SQL FETCH stu INTO :name, :average;
    if (average >= 90) {
        cout << name << " gets an A" << endl;
    }
}
EXEC SQL CLOSE stu;
```

Module languages

many databases provide an alternate, high-level API

- allows user-defined SQL modules or scripts to be called from applications
- scripts are written in an SQL style but with a vendor-specific syntax

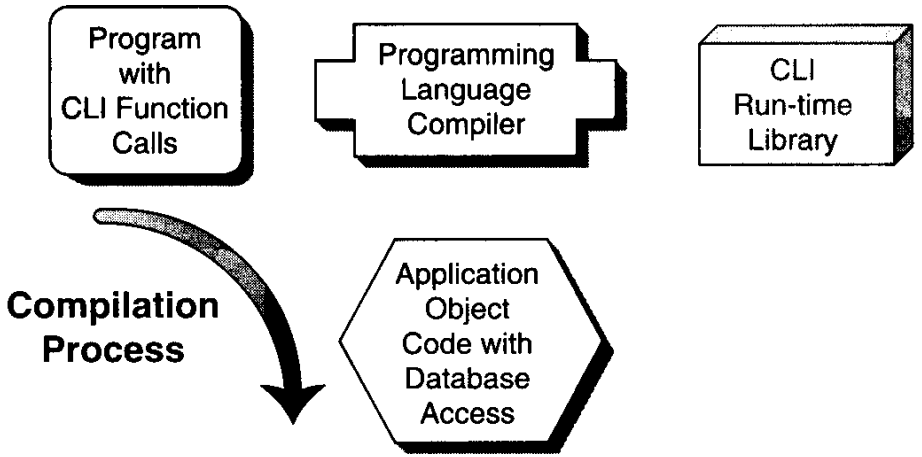
- SQL-92 defined a generic Module Language



Call level interfaces (CLIs)

a lower-level API is the actual SQL library interface used by the preprocessor in translating embedded statements

- can call these library routines directly from an application program
- more complex, but greater flexibility (if you know what you are doing)



Microsoft's version of a CLI is *Open Database Connectivity (ODBC)*

- defines over 80 functions for opening a connection to a database, executing SQL commands, managing transactions, processing data, …

API tradeoffs

embedded SQL

+ direct SQL syntax appears inline with the code (thus clear, familiar)
+ oldest type of API with most support for standard
– many vendor-specific extensions are implemented
– requires a preprocessor

module language

+ pure SQL programming
– syntax can be cumbersome, vendor-specific
– must keep calling code in sync with modules
– requires a module language compiler

call level interfaces

+ direct bindings to SQL libraries without a preprocessor
– low-level interface to SQL environment
– standard is immature and incomplete, not well-supported

4GL alternatives

a variety of fourth-generation development tools and packaged applications can be used as alternatives to 3GL APIs

- GUI generators *e.g., Visual Basic*
  4GL tools allow user to specify interface, code automatically generated
  3GL code can be entered into stubs to complete applications

- application generators *e.g., PowerBuilder*
  similar to GUI generators, but provide English-like scripting language

- application development environments *e.g., Forte*
  extend application generators to generate both client- and server-side code
  also provide integration with CASE tools

- query & report development environments *e.g., SAS*
  limit application generators to only access data, no update capabilities
  as a result, generally smaller, faster, and easier to learn

- packaged end-user applications *e.g., Excel*
  can use an Excel macro to access a relational database via ODBC

Next week…

Relax, eat some tofu turkey, …

Next next week…

Networked SQL

- formats and protocols (FAPs), FAP gateways
- distributed aspects
- performance aspects
- online analytical processing aspects

Read Chapter 13

As always, be prepared for a short quiz