SE351: Fundamentals of Database Systems

SQL Programming

Jing Sun and Miao Qiao
The University of Auckland

Overview

Weeks 1-6:

- Relational Data Model
- Entity-Relationship (ER) Model
- SQL and Relational Algebra
- Functional Dependency and Normal Form

Overview

Weeks 1-6:

- Relational Data Model
- Entity-Relationship (ER) Model
- SQL and Relational Algebra
- Functional Dependency and Normal Form

Reading material:

- Chapter 10 of the textbook and
- Chapter 9 of book "A First Course in Database Systems".

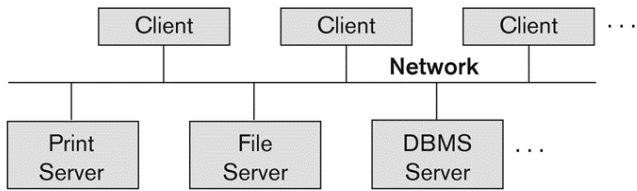
Architectures

- Small and Standalone DB
- Client/Server Architecture

Architectures

- Small and Standalone DB
- Client/Server Architecture
  - Client: A user machine that provide user interface and local processing
  - Server: A system with hardware and software that can provide services to the client machines
    - Specialized Servers with Specialized functions
    - Clients can access the specialized servers as needed

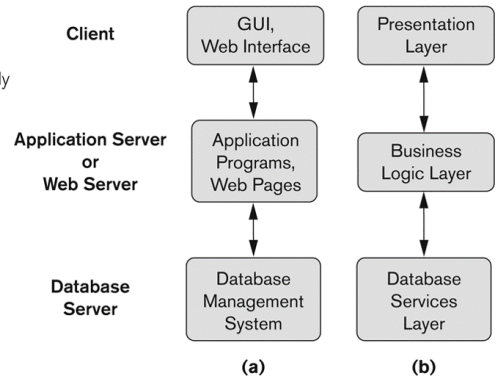
Figure 2.5
Logical two-tier client/server architecture.



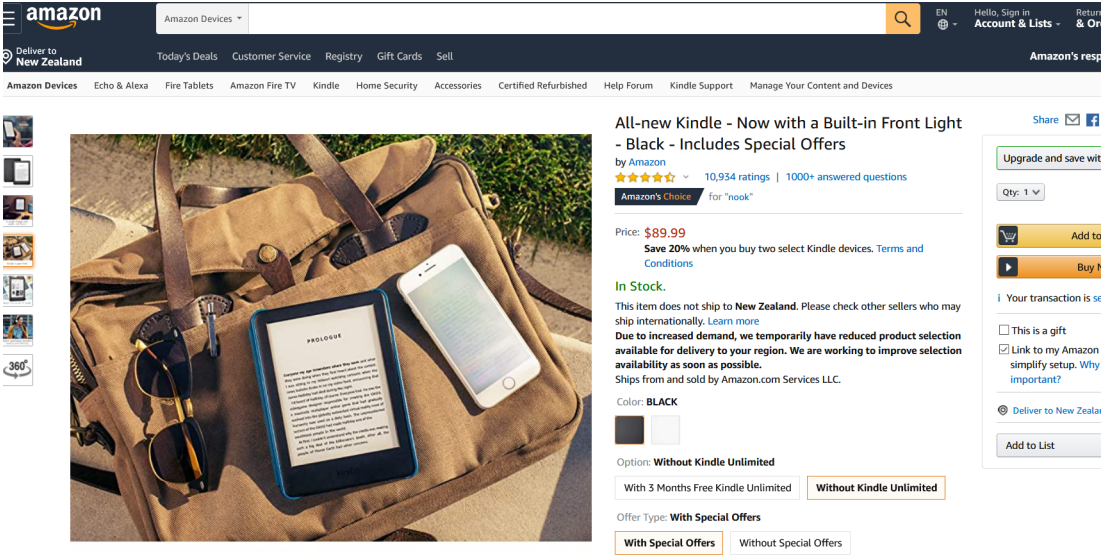
Architectures

- Small, Standalone DB
- Client/Server Architecture
- Logical Three-tier Architecture
- ...

Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.



Web Server, Application Server and Database Server



Problems of This Week

- How ordinary programs interact with DB?
- How to resolve *impedance mismatch* and then process query results?
- How to properly manage the connections?

SQL Environment

SQL Environment
: a framework under which data may exist and SQL on data may be executed.

- Schema: A collection of tables, views, assertions, triggers, and etc..
- Catalog: A collection of schemas, each schema has a unique name. (INFORMATION_SCHEMA)
- Clusters: A collection of catalogs, one user has only one cluster – the DB of the user.

Name for schema elements: CatalogName.SchemaName.TableName
e.g., MovieCatalog.MovieSchema.Movies

SQL Environment

- Connection: connects between an SQL-client and an SQL-server with an SQL statement

```
CONNECT TO <serve name> AS <connection name>
AUTHORIZATION <name and password>
SET CONNECTION conn1;
DISCONNECT conn1;
```

SQL Environment

- Connections
- Sessions: the SQL between the the starting and the termination of a connection.

SQL Environment

- Connections
- Sessions
- Modules: the SQL used by an application program.
  - Generic SQL Interface: we sit at a terminal and ask queries of a database.
  - Real SQL programming: conventional programs interacting with SQL
    - Code in a specialized language that is stored in the database itself (e.g., PSM, PL/SQL).
    - SQL statements that are embedded in a *host language* (e.g., C).
    - Connection tools are used to allow a conventional language to access a database (e.g., CLI, JDBC, PHP/DB).

SQL Environment

- Connection
- Session
- Module: the SQL terms used by an application program.

The relationship between an *SQL module* and an *SQL agent*: similar to

- program and process
- class and object

Modules: SQL Programming for Applications

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

Impedance mismatch: the fact that the data model of SQL differs so much from the models of other languages.

- Code in a specialized language is stored in the database itself (e.g., PSM, PL/SQL).
- SQL statements are embedded in a host language (e.g., C).
- Connection tools are used to allow a conventional language to access a database (e.g., CLI, JDBC, PHP/DB).

Stored Procedures

- PSM, or "persistent stored modules," allows us to store procedures as database schema elements.
- PSM = a mixture of conventional statements (if, while, etc.) and SQL.
- Lets us do things we cannot do in SQL alone.

Basic PSM Form

Procedure:

```
CREATE PROCEDURE <name> (<parameter list>)
<optional local declarations>
<body>;
```

Function alternative:

```
CREATE FUNCTION <name> (<parameter list> ) RETURNS <type>
<optional local declarations>
<body>;
```

Parameters in PSM

Unlike the usual name-type pairs in languages like Java, PSM uses mode-name-type triples, where the mode can be:

- IN = procedure uses value, does not change value.
- OUT = procedure changes, does not use.
- INOUT = both.

Example: Stored Procedure

- Let's write a procedure that takes two arguments b and p, and adds a tuple to relation Sells(bar, beer, price) that has bar = 'Joe"s Bar', beer = b, and price = p.

```
CREATE PROCEDURE JoeMenu (
IN b CHAR(20),
IN p  REAL
)
INSERT INTO Sells
VALUES('Joe"s Bar', b, p);
```

- Parameters are both read only, can not be changed.
- Body includes a single insertion.

Invoking Procedures

- Use SQL/PSM statement CALL, with the name of the desired procedure and arguments.
- Example:

```
CALL JoeMenu('Moosedrool', 5.00);
```

- Functions used in SQL expressions wherever a value of their return type is appropriate.

Kinds of PSM Statements

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

- RETURN <expression>: sets the return value of a function.
  Unlike Java, etc., RETURN does not terminate function execution.
- DECLARE <name> <type>: used to declare local variables.
- BEGIN . . . END: for groups of statements, separate statements by semicolons.
- SET <variable> = <expression>; : Assignment statements, e.g., SET b = 'Bud';
- Statement labels: give a statement a label by prefixing a name and a colon, e.g., IF and loop.

IF Statements

- Simplest form:

  `IF <condition> THEN <statements(s)> END IF;`

- Add ELSE <statement(s)> if desired, as

  `IF . . . THEN . . . ELSE . . . END IF;`

- Add additional, if desired, ELSEIF <statements(s)>:

  `IF ... THEN ... ELSEIF ... THEN ...`
  `ELSEIF ... THEN ... ELSE ... END IF;`

Example: IF

- Let's rate bars by how many customers they have, based on Frequents(drinker,bar).
  - $<$ 100 customers: "unpopular".
  - 100-199 customers: "average".
  - $\geq$ 200 customers: "popular".
- Function Rate(b) rates bar b.

```
CREATE FUNCTION Rate (IN b CHAR(20) )
RETURNS CHAR(10)
DECLARE cust INTEGER;
BEGIN
SET cust = (SELECT COUNT(*) FROM Frequents WHERE bar = b);
IF cust < 100 THEN RETURN 'unpopular'
ELSEIF cust < 200 THEN RETURN 'average'
ELSE RETURN 'popular'
END IF;
END;
```

Loops

- Basic form: <loop name>: LOOP <statements> END LOOP;
- Exit from a loop by: LEAVE <loop name>; Example of exiting a loop:

```
loop1: LOOP
. . .
IF ... THEN ... LEAVE loop1;
. . .
END LOOP;
```

- Other Loop Forms

```
WHILE <condition> DO
<statements>
END WHILE;
REPEAT
<statements>
UNTIL <condition> END REPEAT;
```

Queries

- General SELECT-FROM-WHERE queries are not permitted in PSM.
- There are three ways to get the effect of a query:
  - Queries producing one value can be the expression in an assignment.
  - Single-row SELECT . . . INTO.
  - Cursors.

Queries that produce one tuple

- Using local variables and assignment statements:

```
SET p = (SELECT price FROM Sells
WHERE bar = 'Joe"s Bar' AND beer = 'Bud');
```

- Another way to get the value of a query that returns one tuple is by placing INTO <variable> after the SELECT clause.

```
SELECT price INTO p FROM Sells
WHERE bar = 'Joe"s Bar' AND beer = 'Bud';
```

Cursors

- Cursor: a tuple-variable that ranges over all tuples in the result of some query.
- Declare a cursor c by:

```
DECLARE c CURSOR FOR <query>;
```

- Opening and Closing Cursors
  - To use cursor c, we must issue the command:

```
OPEN c;
```

    The query of c is evaluated, and c is set to point to the first tuple of the result.
  - When finished with c, issue command:

```
CLOSE c;
```

Cursors

- Fetching Tuples From a Cursor
  - To get the next tuple from cursor c, issue command:

    ```
    FETCH FROM c INTO x1, x2,...,xn ;
    ```

    The x 's are a list of variables, one for each component of the tuples referred to by c. c is moved automatically to the next tuple.
- The usual way to use a cursor is to create a loop with a FETCH statement, and do something with each tuple fetched.
- A tricky point is how we get out of the loop when the cursor has no more tuples to deliver.

Loops

- Each SQL operation returns a status, which is a 5-digit character string.
  For example, 00000 = "Everything OK," and 02000 = "Failed to find a tuple."
- In PSM, we can get the value of the status in a variable called SQLSTATE.
- We may declare a condition NotFound which is a boolean variable that is true if and only if SQLSTATE has value 02000.

```
DECLARE NotFound CONDITION FOR SQLSTATE '02000';
```

- The structure of a cursor loop is thus:

```
cursorLoop: LOOP
...
FETCH c INTO ... ;
IF NotFound THEN LEAVE cursorLoop;
END IF;
...
END LOOP;
```

Example: Cursor

- Let's write a procedure using cursor that examines Sells(bar, beer, price), and raises by 1 the price of all beers at Joe's Bar that are under 3.

```
CREATE PROCEDURE JoeGouge( )
DECLARE theBeer CHAR(20);
DECLARE thePrice REAL;
DECLARE NotFound CONDITION FOR SQLSTATE '02000';
DECLARE c CURSOR FOR
(SELECT beer, price FROM Sells WHERE bar = 'Joe''s Bar');
BEGIN
OPEN c;
menuLoop: LOOP
FETCH c INTO theBeer, thePrice;
IF NotFound THEN LEAVE menuLoop END IF;
IF thePrice < 3.00 THEN
UPDATE Sells SET price = thePrice + 1.00
WHERE bar = 'Joe''s Bar' AND beer = theBeer;
END IF;
END LOOP; CLOSE c;
END;
```

For Cursor Loop

- In PSM, for loop is used only to iterate over a cursor.

```
FOR < loopname > AS <cursor name> CURSOR FOR <query> DO
<statement list>
END FOR;
```

- Rewrite the previous example:

```
CREATE PROCEDURE JoeGouge( )
BEGIN
FOR barloop AS c CURSOR FOR
SELECT beer, price FROM Sells WHERE bar = 'Joe''s Bar';
DO IF price < 3.00 THEN
UPDATE Sells SET price = price + 1.00 WHERE CURRENT OF c;
END IF;
END FOR;
END;
```

Modules: SQL Programming for Applications

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

- Code in a specialized language is stored in the database itself (e.g., PSM, PL/SQL).
- SQL statements are embedded in a host language (e.g., C).
- Connection tools are used to allow a conventional language to access a database (e.g., CLI, JDBC, PHP/DB).

Modules: SQL Programming for Applications

- Code in a specialized language is stored in the database itself (e.g., PSM, PL/SQL).
- SQL statements are embedded in a host language (e.g., C).
- Connection tools are used to allow a conventional language to access a database (e.g., CLI, JDBC, PHP/DB).

Use conventional languages is to use library calls.

- C + CLI
- Java + JDBC
- PHP + PEAR/DB

JDBC

- Java Database Connectivity (JDBC) is a library with Java as the host language.
- Making a Connection

```
import java.sql.*;
Class.forName(com.mysql.jdbc.Driver);
Connection myCon =
DriverManager.getConnection(DB URL, username, and password);
```

JDBC

JDBC provides two classes

- Statement = an object that can accept a string that is a SQL statement and can execute such a string.
- PreparedStatement = an object that has an associated SQL statement ready to execute.

Creating Statements

```
Statement stat1 = myCon.createStatement();
PreparedStatement stat2 =
myCon.createStatement("SELECT beer, "+
  "price FROM Sells WHERE bar = 'Joe' 's Bar' ");
```

JDBC

Executing SQL Statements

- JDBC distinguishes queries from modifications, which it calls "updates."
- Statement and PreparedStatement each have methods executeQuery and executeUpdate.
- For Statements: one argument: the query or modification to be executed.
- For PreparedStatements: no argument.

```
stat1.executeUpdate("INSERT INTO Sells VALUES('Brass Rail','Bud',3.00)");
```

JDBC

Example: Query

- stat2 is a PreparedStatement holding the query "SELECT beer, price FROM Sells WHERE bar = 'Joe"s Bar' ".
- executeQuery returns an object of class ResultSet.
- The query: ResultSet menu = stat2.executeQuery();
- An object of type ResultSet is something like a cursor.
  - Method next() advances the "cursor" to the next tuple.
  - The first time next() is applied, it gets the first tuple.
  - If there are no more tuples, next() returns the value false.

Accessing Components of Tuples

- When a ResultSet is referring to a tuple, we can get the components of that tuple by applying certain methods to the ResultSet. Method $getX(i)$, where X is some type, and i is the component number, returns the value of that component. The value must have type X.
- Example: Menu = ResultSet for query "SELECT beer, price FROM Sells WHERE bar = 'Joe' 's Bar' ". Access beer and price from each tuple by:

```
while ( menu.next() ) {
theBeer = Menu.getString(1);
thePrice = Menu.getFloat(2);
/*something with theBeer and      thePrice*/
}
```

Impedance mismatch: the fact that the data model of SQL differs so much from the models of other languages.

- Code in a specialized language is stored in the database itself (e.g., PSM, PL/SQL).
- SQL statements are embedded in a host language (e.g., C).
- Connection tools are used to allow a conventional language to access a database (e.g., CLI, JDBC, PHP/DB).

Thank you!