



Lecture 21 of 42

Intro to Web Databases Discussion: Online DBs

Friday, 13 October 2006

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/va60>

Course web site: <http://www.kddresearch.org/Courses/Fall-2006/CIS560>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Reading for Next Class:

Second half of Chapter 8, Silberschatz *et al.*, 5th edition



Chapter 8: Application Design and Development

- User Interfaces and Tools
- Web Interfaces to Databases
- Web Fundamentals
- Servlets and JSP
- Building Large Web Applications
- Triggers
- Authorization in SQL
- Application Security





Example Servlet Code

```
Public class BankQuery(Servlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
    result)
        throws ServletException, IOException {
        String type = request.getParameter("type");
        String number = request.getParameter("number");
        ...code to find the loan amount/account balance ...
        ...using JDBC to communicate with the database..
        ...we assume the value is stored in the variable balance
        result.setContentType("text/html");
        PrintWriter out = result.getWriter( );
        out.println("<HEAD><TITLE>Query Result</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("Balance on " + type + number + "=" + balance);
        out.println("</BODY>");
        out.close ( );
    }
}
```



Server-Side Scripting

- Server-side scripting simplifies the task of connecting a database to the Web
 - * Define a HTML document with embedded executable code/SQL queries.
 - * Input values from HTML forms can be used directly in the embedded code/SQL queries.
 - * When the document is requested, the Web server executes the embedded code/SQL queries to generate the actual HTML document.
- Numerous server-side scripting languages
 - * JSP, Server-side Javascript, ColdFusion Markup Language (cfml), PHP, Jscript
 - * General purpose scripting languages: VBScript, Perl, Python



Improving Web Server Performance

- Performance is an issue for popular Web sites
 - * May be accessed by millions of users every day, thousands of requests per second at peak time
- Caching techniques used to reduce cost of serving pages by exploiting commonalities between requests
 - * At the server site:
 - ⇒ Caching of JDBC connections between servlet requests
 - ⇒ Caching results of database queries
 - ◆ Cached results must be updated if underlying database changes
 - ⇒ Caching of generated HTML
 - * At the client's network
 - ⇒ Caching of pages by Web proxy



Trigger Example

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
 - * setting the account balance to zero
 - * creating a loan in the amount of the overdraft
 - * giving this loan a loan number identical to the account number of the overdrawn account
- The condition for executing the trigger is an update to the *account* relation that results in a negative *balance* value.





Trigger Example in SQL:1999

```
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
  insert into borrower
  (select customer-name, account-number
   from depositor
   where nrow.account-number =
         depositor.account-number);
  insert into loan values
  (n.row.account-number, nrow.branch-name,
   - nrow.balance);

  update account set balance = 0
  where account.account-number = nrow.account-number
end
```



Triggering Events and Actions in SQL

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
 - * E.g. **create trigger overdraft-trigger after update of balance on account**
- Values of attributes before and after an update can be referenced
 - * **referencing old row as** : for deletes and updates
 - * **referencing new row as** : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. E.g. convert blanks to null.

```
create trigger setnull-trigger before update on r
referencing new row as nrow
for each row
when nrow.phone-number = ''
set nrow.phone-number = null
```





Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
 - * Use **for each statement** instead of **for each row**
 - * Use **referencing old table** or **referencing new table** to refer to temporary tables (called **transition tables**) containing the affected rows
 - * Can be more efficient when dealing with SQL statements that update a large number of rows



External World Actions

- We sometimes require external world actions to be triggered on a database update
 - * E.g. re-ordering an item whose quantity in a warehouse has become small, or turning on an alarm light,
- Triggers cannot be used to directly implement external-world actions, BUT
 - * Triggers can be used to record actions-to-be-taken in a separate table
 - * Have an external process that repeatedly scans the table, carries out external-world actions and deletes action from table
- E.g. Suppose a warehouse has the following tables
 - * *inventory (item, level)*: How much of each item is in the warehouse
 - * *minlevel (item, level)*: What is the minimum desired level of each item
 - * *reorder (item, amount)*: What quantity should we re-order at a time
 - * *orders (item, amount)*: Orders to be placed (read by external process)





External World Actions (Cont.)

```
create trigger reorder-trigger after update of amount on inventory
referencing old row as orow, new row as nrow
for each row
  when nrow.level <= (select level
                      from minlevel
                      where minlevel.item = orow.item)
  and orow.level > (select level
                    from minlevel
                    where minlevel.item = orow.item)
begin
  insert into orders
  (select item, amount
   from reorder
   where reorder.item = orow.item)
end
```



Triggers in MS-SQLServer Syntax

```
create trigger overdraft-trigger on account
for update
as
if inserted.balance < 0
begin
  insert into borrower
  (select customer-name, account-number
   from depositor, inserted
   where inserted.account-number =
         depositor.account-number)
  insert into loan values
  (inserted.account-number, inserted.branch-name,
   - inserted.balance)
  update account set balance = 0
  from account, inserted
  where account.account-number = inserted.account-number
end
```



When Not To Use Triggers

- Triggers were used earlier for tasks such as
 - * maintaining summary data (e.g. total salary of each department)
 - * Replicating databases by recording changes to special relations (called **change** or **delta** relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
 - * Databases today provide built in materialized view facilities to maintain summary data
 - * Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
 - * Define methods to update fields
 - * Carry out actions as part of the update methods instead of through a trigger



Authorization in SQL (see also Section 4.3)

Forms of authorization on parts of the database:

- **Read authorization** - allows reading, but not modification of data.
- **Insert authorization** - allows insertion of new data, but not modification of existing data.
- **Update authorization** - allows modification, but not deletion of data.
- **Delete authorization** - allows deletion of data





Authorization (Cont.)

Forms of authorization to modify the database schema:

- **Index authorization** - allows creation and deletion of indices.
- **Resources authorization** - allows creation of new relations.
- **Alteration authorization** - allows addition or deletion of attributes in a relation.
- **Drop authorization** - allows deletion of relations.



Authorization and Views

- Users can be given authorization on views, without being given any authorization on the relations used in the view definition
- Ability of views to hide data serves both to simplify usage of the system and to enhance security by allowing users access only to data they need for their job
- A combination of relational-level security and view-level security can be used to limit a user's access to precisely the data that user needs.





View Example

- Suppose a bank clerk needs to know the names of the customers of each branch, but is not authorized to see specific loan information.
 - * Approach: Deny direct access to the *loan* relation, but grant access to the view *cust-loan*, which consists only of the names of customers and the branches at which they have a loan.
 - * The *cust-loan* view is defined in SQL as follows:

```
create view cust-loan as  
  select branchname, customer-name  
  from borrower, loan  
  where borrower.loan-number = loan.loan-number
```



View Example (Cont.)

- The clerk is authorized to see the result of the query:

```
select *  
from cust-loan
```
- When the query processor translates the result into a query on the actual relations in the database, we obtain a query on *borrower* and *loan*.
- Authorization must be checked on the clerk's query before query processing replaces a view by the definition of the view.



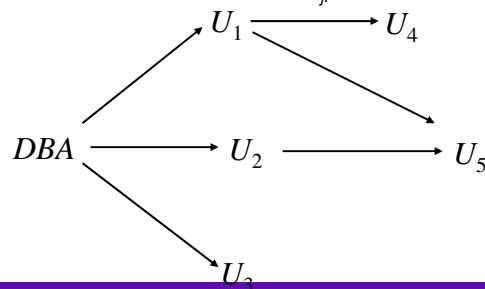
Authorization on Views

- Creation of view does not require **resources** authorization since no real relation is being created
- The creator of a view gets only those privileges that provide no additional authorization beyond that he already had.
- E.g. if creator of view *cust-loan* had only **read** authorization on *borrower* and *loan*, he gets only **read** authorization on *cust-loan*



Granting of Privileges

- The passage of authorization from one user to another may be represented by an authorization graph.
- The nodes of this graph are the users.
- The root of the graph is the database administrator.
- Consider graph for update authorization on loan.
- An edge $U_i \rightarrow U_j$ indicates that user U_i has granted update authorization on loan to U_j .





Authorization Grant Graph

- *Requirement:* All edges in an authorization graph must be part of some path originating with the database administrator
- If DBA revokes grant from U_1 :
 - * Grant must be revoked from U_4 since U_1 no longer has authorization
 - * Grant must not be revoked from U_5 since U_5 has another authorization path from DBA through U_2
- Must prevent cycles of grants with no path from the root:
 - * DBA grants authorization to U_7
 - * U_7 grants authorization to U_8
 - * U_8 grants authorization to U_7
 - * DBA revokes authorization from U_7
- Must revoke grant U_7 to U_8 and from U_8 to U_7 since there is no path from DBA to U_7 or to U_8 anymore.



Security Specification in SQL

- The grant statement is used to confer authorization
grant <privilege list>
on <relation name or view name> to <user list>
- <user list> is:
 - * a user-id
 - * *public*, which allows all valid users the privilege granted
 - * A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).





Privileges in SQL

- **select**: allows read access to relation, or the ability to query using the view
 - * Example: grant users U_1 , U_2 , and U_3 **select** authorization on the *branch* relation:
grant select on branch to U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **references**: ability to declare foreign keys when creating relations.
- **usage**: In SQL-92; authorizes a user to use a specified domain
- **all privileges**: used as a short form for all the allowable privileges



Privilege To Grant Privileges

- **with grant option**: allows a user who is granted a privilege to pass the privilege on to other users.
 - * Example:
grant select on branch to U_1 with grant option
gives U_1 the **select** privileges on branch and allows U_1 to grant this privilege to others





Roles

- Roles permit common privileges for a class of users can be specified just once by creating a corresponding “role”
- Privileges can be granted to or revoked from roles, just like user
- Roles can be assigned to users, and even to other roles
- SQL:1999 supports roles

```
create role teller
create role manager
```

```
grant select on branch to teller
grant update (balance) on account to teller
grant all privileges on account to manager
```

```
grant teller to manager
```

```
grant teller to alice, bob
grant manager to avi
```



Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.
revoke<privilege list>
on <relation name or view name> **from** <user list> [**restrict**]**cascade**]
- Example:
revoke select on branch from U₁, U₂, U₃ cascade
- Revocation of a privilege from a user may cause other users also to lose that privilege; referred to as cascading of the **revoke**.
- We can prevent cascading by specifying **restrict**:
revoke select on branch from U₁, U₂, U₃ restrict
With **restrict**, the **revoke** command fails if cascading revokes are required.





Revoking Authorization in SQL (Cont.)

- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- If <revokee-list> includes **public** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.



Limitations of SQL Authorization

- SQL does not support authorization at a tuple level
 - * E.g. we cannot restrict students to see only (the tuples storing) their own grades
- With the growth in Web access to databases, database accesses come primarily from application servers.
 - * End users don't have database user ids, they are all mapped to the same database user id
- All end-users of an application (such as a web application) may be mapped to a single database user
- The task of authorization in above cases falls on the application program, with no support from SQL
 - * Benefit: fine grained authorizations, such as to individual tuples, can be implemented by the application.
 - * Drawback: Authorization must be done in application code, and may be dispersed all over an application
 - * Checking for absence of authorization loopholes becomes very difficult since it requires reading large amounts of application code





Audit Trails

- An audit trail is a log of all changes (inserts/deletes/updates) to the database along with information such as which user performed the change, and when the change was performed.
- Used to track erroneous/fraudulent updates.
- Can be implemented using triggers, but many database systems provide direct support.



Application Security

- Data may be *encrypted* when database authorization provisions do not offer sufficient protection.
- Properties of good encryption technique:
 - * Relatively simple for authorized users to encrypt and decrypt data.
 - * Encryption scheme depends not on the secrecy of the algorithm but on the secrecy of a parameter of the algorithm called the encryption key.
 - * Extremely difficult for an intruder to determine the encryption key.





Encryption (Cont.)

- *Data Encryption Standard (DES)* substitutes characters and rearranges their order on the basis of an encryption key which is provided to authorized users via a secure mechanism. Scheme is no more secure than the key transmission mechanism since the key has to be shared.
 - *Advanced Encryption Standard (AES)* is a new standard replacing DES, and is based on the Rijndael algorithm, but is also dependent on shared secret keys
 - *Public-key encryption* is based on each user having two keys:
 - * *public key* – publicly published key used to encrypt data, but cannot be used to decrypt data
 - * *private key* -- key known only to individual user, and used to decrypt data. Need not be transmitted to the site doing encryption.
- Encryption scheme is such that it is impossible or extremely hard to decrypt data given only the public key.
- The *RSA* public-key encryption scheme is based on the hardness of factoring a very large number (100's of digits) into its prime components.



Authentication

- Password based authentication is widely used, but is susceptible to sniffing on a network
- **Challenge-response** systems avoid transmission of passwords
 - * DB sends a (randomly generated) challenge string to user
 - * User encrypts string and returns result.
 - * DB verifies identity by decrypting result
 - * Can use public-key encryption system by DB sending a message encrypted using user's public key, and user decrypting and sending the message back
- **Digital signatures** are used to verify authenticity of data
 - * E.g. use private key (in reverse) to encrypt data, and anyone can verify authenticity by using public key (in reverse) to decrypt data. Only holder of private key could have created the encrypted data.
 - * Digital signatures also help ensure **nonrepudiation**: sender cannot later claim to have not created the data





Digital Certificates

- **Digital certificates** are used to verify authenticity of public keys.
- Problem: when you communicate with a web site, how do you know if you are talking with the genuine web site or an imposter?
 - * Solution: use the public key of the web site
 - * Problem: how to verify if the public key itself is genuine?
- Solution:
 - * Every client (e.g. browser) has public keys of a few root-level **certification authorities**
 - * A site can get its name/URL and public key signed by a certification authority: signed document is called a **certificate**
 - * Client can use public key of certification authority to verify certificate
 - * Multiple levels of certification authorities can exist. Each certification authority
 - ⇒ presents its own public-key certificate signed by a higher level authority, and
 - ⇒ Uses its private key to sign the certificate of other web sites/authorities

