BKM_DATS: Databázové systémy

12. Securing Database

Vlastislav Dohnal

Credits

- Materials are based on presentations:
  - Courses CS245, CS345, CS345
    - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
    - Stanford University, California
  - Course CS145 following the book
    - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom: Database Systems: The Complete Book
  - Book
    - Andrew J. Brust, Stephen Forte: Mistrovství v programování SQL Serveru 2005
  - MSDN library by Microsoft
  - Hasura.io

Contents

- **DB security**
  - **Access control in DB**
  - Stored procedures
  - Attack on DB

Access Control – Authorization

- Analogy to file systems
  - Objects
    - File, directory, …
  - Subject
    - Typically: owner, group, others (all users)
  - Access Right
    - Defined on an object *O* for a subject *S*
    - Typically: read, write, execute

Privileges

- Database systems
  - Typically, finer granularity than the typical file system
  - Varies for objects
    - Tables, views, sequences, schema, database, procedures, …
  - Views
    - an important tool for access control
  - Subjects are typically user and group
    - Often referred as *authorization id* or *role*
    - Subject “others“ is denoted as PUBLIC
      - Granting access for PUBLIC means allowing access to anyone.

Privileges

- For relations/tables:
  - SELECT
    - query the table's content (i.e. print rows)
    - Sometimes can be limited to selects attributes
  - INSERT
    - Sometimes can be limited to selects attributes
  - DELETE
  - UPDATE
    - Sometimes can be limited to selects attributes
  - REFERENCES
    - creating foreign keys referencing this table

Privileges

- Example
  - INSERT INTO Beers(name)

```
SELECT beer FROM Sells
WHERE NOT EXISTS
        (SELECT * FROM Beers
          WHERE name = beer);
```

We add beers that do not appear in Beers; leaving manufacturer NULL.

  - Requirements for privileges:
    - INSERT on the table *Beers*
    - SELECT on *Sells* and *Beers*

Privileges

- Views as Access Control
  - Relation
    - Employee(id, name, address, salary)
  - Want to make salary confidential:
    - CREATE VIEW EmpAddress AS
      SELECT id, name, address
      FROM Employee;
    - Privileges:
      - Revoke SELECT from table Employee
      - Grant SELECT on EmpAddress

Privileges

- Granting privileges
  - GRANT <list of privileges>
    ON <relation or object>
    TO <list of authorization ID's>;
- You may also grant "grant privilege"
  - By appending clause "WITH GRANT OPTION"
    - GRANT SELECT
      ON TABLE EmpAddress
      TO karel
      WITH GRANT OPTION

Privileges

- Example (to be run as owner of *sells*)
  - GRANT SELECT, UPDATE(price)
    ON sells TO sally;
- User *sally* can
  - Read (select) from table *sells*
  - Update values in attribute *price*

Privileges

- Example (to be run as owner of *sells*)
  - GRANT UPDATE ON sells TO sally
    WITH GRANT OPTION;
- User *sally* can
  - Update values of any attribute in *sells*
  - Grant access to other users
    - Only UPDATE can be granted, but can be limited to some attributes.

Privileges

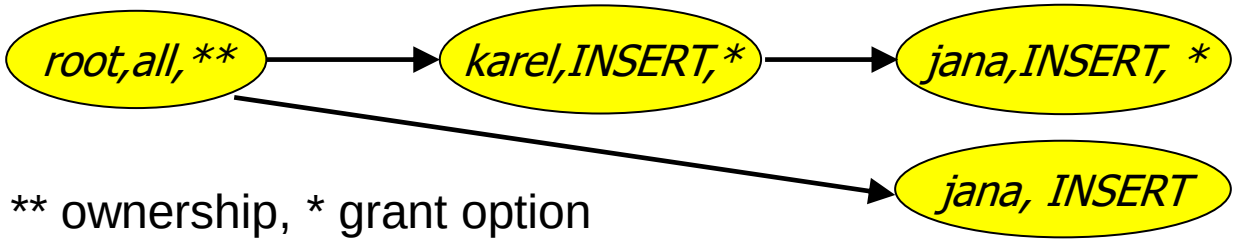
- Revoking statement
  - REVOKE <list of privileges>
    ON <relation or object>
    FROM <list of authorization ID's>;
- Listed users can no longer use the priviledges.
  - But they may still have the privilege
  - → because they obtained it independently from elsewhere.
    - Or they are members of a group or PUBLIC is applied

Privileges

- Revoking privileges
  - Appending to REVOKE statement:
    - CASCADE – Now, any grants made by a revokee are also not in force, no matter how far the privilege was passed
    - RESTRICT (implicit) –
      - If the privilege has been passed to others, the REVOKE fails as a warning
      - So something else must be done to “chase the privilege down.”
  - REVOKE GRANT OPTION FOR …
    - Removes the “grant option” only.
    - Omitting this leads to removing the privilege and also the grant option!

Privileges – Diagram

- Diagram depict privileges granted by a grantor to a grantee



  - Each object has its diagram
  - Node is specified by
    - Role (user / group)
    - Granted privilege
    - Flag of ownership or granting option
  - Edge from X to Y
    - X has granted the privilege to Y

Privileges – Diagram

- „*root,all*“ denotes
  - user *root* has privilege *all*.
- Privilege „*all*“ on table means
  - = insert, update, delete, select, references
- Grant option “*“
  - The privilege can by granted by the user
- Option “**“
  - Object owner (root node of each diagram)
- Object owner
  - All is granted by default
  - Can pass the privileges to other users

Creating user accounts

- Add a new account

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

Connections to DB server

- config_file (postgresql.conf)
  - max_connections
  - ssl
- hba_file (pg_hba.conf)
  - Configures client authentication
    - source address, database, username

```
local      database  user  auth-method [auth-options]
host       database  user  address     auth-method  [auth-options]
```

Connections to DB server

- hba_file example

```
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication     postgres                                peer
#host    replication     postgres        127.0.0.1/32            md5

host    all         all         147.251.50.0/24             password
host    lectures    PB154       0.0.0.0/0               password
host    all         PB154       0.0.0.0/0               reject
host    all         all         84.242.71.236/32        trust
```

Client connecting to DB

- Need to specify where to connect
  - postgresql://username:password@host:port/dbname[?paramspec]
  - E.g., postgresql://karel:pwd@db.fi.muni.cz:5432/pgdb
- Parameters
  - Format: …/dbname?name=value&name2=v2
  - ssl, user, password, options
  - E.g., options=-c search_path=test,public

Implementation of clients

- JDBC / ODBC
  - General interface for connecting & executing queries
- Functions in programming languages
  - Similar to JDBC
- Frameworks
  - Spring.io
  - Hasura.io

Contents

- DB security
  - Access control in DB
  - **Stored procedures**
  - Attack on DB

Stored Procedures

- User-defined program implementing an activity
  - E.g., factorial computation, distance between GPS coords, inserting rows to multiple tables, …
- PostgreSQL
  - CREATE FUNCTION name ([parameters,…]) [RETURNS type]
    …code…

Stored Procedures

- Example:
  - Compute average salary without revealing the individual salaries
    - Table Employee(id, name, address, salary)
  - PostgreSQL:
    - CREATE FUNCTION avgsal() RETURNS real AS ‘SELECT avg(salary) FROM employee’ LANGUAGE SQL;
  - User executes the procedure (function):
    - SELECT avgsal();

Stored Procedures

- Example (cont.):
  - Salaries are not *secured*
  - To secure we need to
    - REVOKE SELECT ON Employee FROM …
    - GRANT EXECUTE ON FUNCTION avgsal() TO …
  - By running “SELECT avgsal();” the procedure is executed with privileges of current user.
  - → it needs SELECT on Employee!

Stored Procedures

- Context of execution
  - Can be set during procedure creation
  - Types:
    - **INVOKER** – run in the context of user that calls the function (typically current user)
    - **DEFINER**– run in the context of the owner of function
    - **„particular user“** – run in the context of the selected user
    - **…**

Stored Procedures

- Execution context
  - PostgreSQL
    - SECURITY INVOKER
    - SECURITY DEFINER
- Solution: set the context to owner
  - CREATE FUNCTION …. LANGUAGE SQL **SECURITY DEFINER**;
    - Assumption: owner has the SELECT privilege to Employee

Contents

- DB security
  - Access control in DB
  - Stored procedures
  - **Attack on DB**

Attacks to DB system

- Network connection
  - DB port open to anyone → use firewall
  - Unsecured connection
    - Apply SSL
- Logging in
  - Weak password
  - Limit users to logging in
    - Allow selected user accounts, IP addresses and databases
  - Using one generic (admin) DB account

Attacks to DB system

- SQL injection
  - Attack by sending SQL commands in place of valid data in forms.
  - Typically related to using only one DB account
    - which is admin)-:

SQL injection – example

- App presents a form to enter string to update customer's note in DB:
  - Internally the app use the following DB statement:

```
UPDATE customer SET note='$note'
WHERE id=current_user;
```

- Malicious user enters to the form:

```
Vader'; DROP TABLE customer; --
```

- After variable expansion we get string:

```
UPDATE customer
SET note='Vader'; DROP TABLE customer; --'
WHERE id=current_user;
```

All in one line!

SQL Injection: Countermeasures

- Use specific user account
  - Avoid using admin account
- Check input values
  - Input length, escape characters,…
- Functions in programming language
  - *mysql_real_escape_string()*, *add_slashes()*
  - $dbh->*quote*($string)
- Functions in DB
  - *quote_literal(str)*
    - returns a string *str* suitably quoted to be used as a string literal in an SQL statement

SQL Injection: Countermeasures

- Prepared statements
  - Parsed statements prepared in DB
    - i.e., compiled templates ready for use
  - Values are then substituted
    - Parameters do not need to be quoted then
  - May be used repetitively
  - Example:

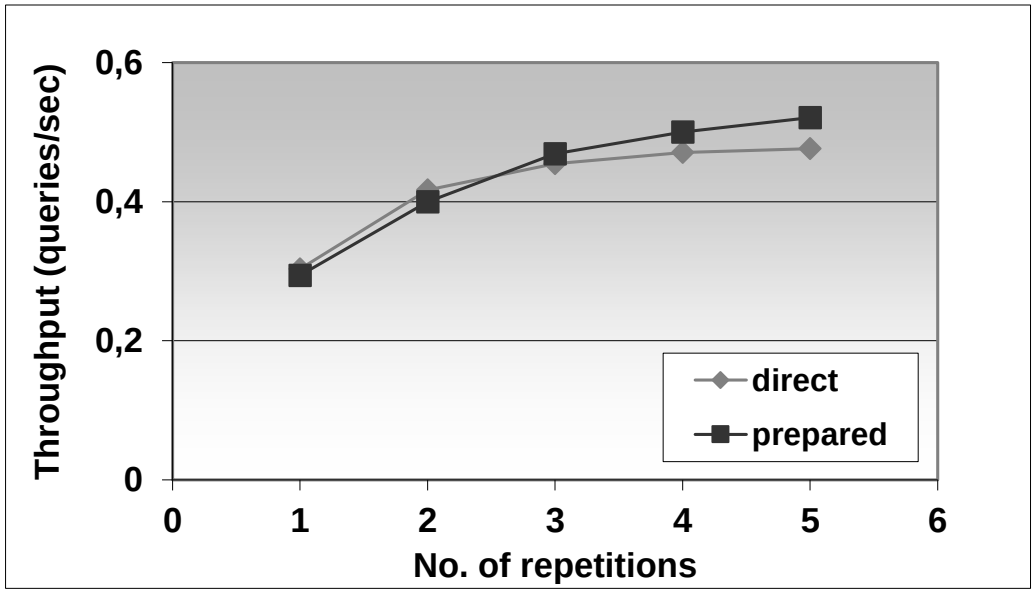
```
$st = $dbh->prepare("SELECT * FROM emp WHERE name LIKE ?");
$st->execute(array( "%$_GET[name]%" ));
```

SQL Injection: Countermeasures

- Prepared statements at server-side
  - The same concept, but stored in DB
  - Typically in procedural languages in DB
  - PostgreSQL
    - PREPARE emp_row(text) AS SELECT * FROM emp WHERE name LIKE **$1**;
      EXECUTE emp_row(**'%John%'**);
- Query is planned in advance
  - Planning time can be amortized
  - But: the plan is generic!
    - i.e., without any optimization induced by knowing the parameter
  - Lasts only for the duration of the current db session

Prepared Statements: Performance

- Prepared execution yields better performance when the query is executed more than once:
  - No compilation
  - No access to catalog.



- Experiment performed on Oracle8iEE on Windows 2000.

Attacking Views

- Views protect data rows…
  - if permissions are correctly set
  - E.g., student(studentid, firstname, lastname, fieldofstudy)
    - CREATE OR REPLACE VIEW studentssme AS SELECT * FROM student WHERE fieldofstudy = 'N-SSME';
  - But, creating a “cheap” function
    - CREATE OR REPLACE FUNCTION test(name text, study text) RETURNS boolean AS $$
      begin
      raise notice 'Name: %, Study: %', name, study;
      return true;
      end;
      $$ LANGUAGE plpgsql VOLATILE COST 0.00001;
  - The query leaks other students in a side channel…
    - SELECT * FROM studentssme WHERE test(lastname, fieldofstudy)
      - NOTICE: Name: Nový, Study: N-AplInf
        NOTICE: Name: Dlouhý, Study: N-Inf
        NOTICE: Name: Svoboda, Study: N-AplInf
        NOTICE: Name: Starý, Study: N-SSME
        NOTICE: Name: Lukáš, Study: N-SSME
        …
- Countermeasures:
  - ban creating new DB objects
  - use security_barrier in Pg.conf or in create view

Lecture Takeaways

- Securing DB
  - Avoid using admin account for general use
  - Limit connections using IP addresses
  - Create triggers to automate some actions
  - Use stored functions for complicated updates
  - Check any input value before using it in SQL query