CERN

Szymon Skorupinski

Oracle Database Security for Developers

Outline

- Users
- Privileges
- Roles
- Encryption
- SQL injection

Outline

- Users
- Privileges
- Roles
- Encryption
- SQL injection

Database users at CERN

- Managed through
    - https://cern.ch/account
- More information about account management
    - Previous tutorial
        - http://cern.ch/go/F8m6
    - SNOW knowledge base (accounts, ownership and passwords)
        - https://cern.service-now.com/service-portal/article.do?n=KB0000947
        - https://cern.service-now.com/service-portal/article.do?n=KB0001593
        - https://cern.service-now.com/service-portal/article.do?n=KB0000829

Database user

- Username along with additional set of attributes
  - Password and its status
    - Expired or not
  - Account status
    - Locked or unlocked
  - Authentication method
  - Default tablespaces for permanent and temporary data storage
  - Tablespace quotas
  - Profile

User creation (1/2)

- Implicit creation of schema
  - Logical container for database objects
  - One-to-one relationship with username

```
SYS@DB> CREATE USER orauser1 IDENTIFIED BY "password"
        DEFAULT TABLESPACE users QUOTA 10M ON users
        TEMPORARY TABLESPACE temp
        PROFILE cern_dev_profile
        PASSWORD EXPIRE
        ACCOUNT UNLOCK;
SYS@DB> GRANT create session TO orauser1;
```

User creation (2/2)

```
$ sqlplus orauser1@db

Enter password:
ERROR:
ORA-28001: the password has expired

Changing password for orauser1
New password:
Retype new password:
Password changed

ORAUSER1@DB>
```

Useful user-related views

View name	Describes
`USER_USERS`	Current user
`USER_TS_QUOTAS`	Tablespace quotas
`ALL_OBJECTS`	All objects accessible to the current user
`USER_OBJECTS`	All objects owned by the current user

Passwords

- Since 11g (finally!) passwords are case-sensitive
  - For backward compatibility passwords set in previous version become case-sensitive only after change is done
- Password policies at CERN
  - Minimum 8 characters
  - Cannot be the same as username and too simple
  - Should differ from previous one by at least 3 characters
  - Contains at least 3 of these categories
    - Lowercase letters, uppercase letters, numbers, symbols
      - But avoid usage of # $ / @)
  - Enforced using password verify function which leads us to...

Profiles

- Named sets of limits on database resources and password access to the database, e.g.
  - `SESSIONS_PER_USER`
  - `IDLE_TIME`
  - `FAILED_LOGIN_ATTEMPTS`
  - `PASSWORD_LIFE_TIME`
  - `PASSWORD_REUSE_TIME`
  - `PASSWORD_REUSE_MAX`
  - `PASSWORD_VERIFY_FUNCTION`
  - `PASSWORD_LOCK_TIME`
  - `PASSWORD_GRACE_TIME`

Useful profile-related views

View name	Describes
`USER_PASSWORD_LIMITS`	Password profile parameters that are assigned to the user
`USER_RESOURCE_LIMITS`	Resource limits for the current user

Outline

- Users
- Privileges
- Roles
- Encryption
- SQL injection

Privileges

- Privilege – right to perform particular type of actions or to access database objects
  - System privileges – ability to perform particular action on database or on any object of specific type, e.g.
    - **`CREATE SESSION, DROP ANY TABLE, ALTER ANY PROCEDURE`**
    - And over 100 others more
    - Be careful when using **`WITH ADMIN OPTION`**
  - Object privileges – ability to perform particular action on a specific schema object, e.g.
    - **`SELECT, INSERT, UPDATE, DELETE, EXECUTE`**
    - Be careful when using **`WITH GRANT OPTION`**

System privileges

- Not all objects have dedicated system privileges

```
ORAUSER1@DB> SELECT username, privilege FROM user_sys_privs;
USERNAME                      PRIVILEGE
----------------------------- ----------------------------------------
ORAUSER1                      CREATE TABLE
ORAUSER1                      CREATE SESSION
ORAUSER1@DB> CREATE TABLE todel(id INT);
Table created.
ORAUSER1@DB> CREATE INDEX todel_idx ON todel(id);
Index created.
```

Object privileges (1/2)

- User automatically has all object privileges for schema objects contained in his/her schema
- Different object privileges are available for different types of schema objects, e.g.
    - **EXECUTE** privilege not relevant for tables
- Some objects do not have any associated object privileges, e.g.
    - Indexes, triggers, database links

Object privileges (2/2)

- Shortcut to grant or revoke all privileges possible for specific object
  - Still individual privileges can be revoked

```
ORAUSER1@DB> GRANT ALL ON orauser1.todel TO orauser2;
Grant succeeded.
```

```
ORAUSER2@DB> SELECT owner, table_name, privilege
             FROM user_tab_privs WHERE table_name = 'TODEL';
OWNER       TABLE_NAME                 PRIVILEGE
----------- -------------------------- ----------------------------------------
ORAUSER1    TODEL                      ALTER
ORAUSER1    TODEL                      DELETE
(...)
```

Useful privilege-related views

View name	Describes
[ALL|USER]_COL_PRIVS	Column object grants for which the current [user or PUBLIC|user] is owner, grantor or grantee
[ALL|USER]_COL_PRIVS_MADE	Column object grants for which the current user is object [owner or grantor|owner]
[ALL|USER]_COL_PRIVS_RECD	Column object grants for which the current [user or PUBLIC|user] is grantee
[ALL|USER]_TAB_PRIVS_MADE	Object grants [made by the current user or made on the objects owned by current user|made on the objects owned by current user]
[ALL|USER]_TAB_PRIVS_RECD	Object grants for which the [user or PUBLIC|user] is the grantee
[ALL|USER]_TAB_PRIVS	Grants on objects where the current [user or PUBLIC|user] is grantee
USER_SYS_PRIVS	System privileges granted to the current user
SESSION_PRIVS	Privileges currently enabled for the current user

Outline

- Users
- Privileges
- Roles
- Encryption
- SQL injection

Roles

- Role – named group of related privileges
  - Could be granted to users or to other roles
- Predefined or user-created
  - Since 11g **`CONNECT`** role has only **`CREATE SESSION`** privilege
- Enabled or disabled
  - Default roles are automatically enabled
  - Provide selective availability of privileges
  - Could be password-protected
- **`PUBLIC`** role
  - Be careful - all grants to this role are available to every user

Roles in PL/SQL (1/4)

- Disabled in any named definer's rights PL/SQL block
- Enabled in any anonymous or named invoker's rights PL/SQL block
  - Anonymous blocks always behave like invoker's right ones
- Definer's rights routine – executed with privileges of its owner
  - Default mode when **`AUTHID`** clause not specified
  - Only **`EXECUTE`** privilege needed for other users
- Invoker's rights routine – executed with privileges of the invoking user

Roles in PL/SQL (2/4)

```
ORAUSER1@DB> SELECT * FROM session_roles;
ROLE
------------------------------
CONNECT
ORAUSER1@DB> SET SERVEROUTPUT ON
ORAUSER1@DB> DECLARE
       l_role_name VARCHAR2(100);
     BEGIN
       SELECT role INTO l_role_name FROM session_roles WHERE rownum = 1;
       dbms_output.put_line(CHR(10) || l_role_name);
     END;
     /
CONNECT
PL/SQL procedure successfully completed.
```

Roles in PL/SQL (3/4)

```
ORAUSER1@DB> CREATE OR REPLACE PROCEDURE show_session_roles_definer AS
       l_role_name VARCHAR2(100);
     BEGIN
       SELECT role INTO l_role_name FROM session_roles WHERE rownum = 1;
       dbms_output.put_line(CHR(10) || l_role_name);
     END;
     /
Procedure created.
ORAUSER1@DB> EXEC show_session_roles_definer
BEGIN show_session_roles_definer; END;
*
ERROR at line 1:
ORA-01403: no data found
(...)
```

Roles in PL/SQL (4/4)

```
ORAUSER1@DB> CREATE OR REPLACE PROCEDURE show_session_roles_invoker
     AUTHID CURRENT_USER AS
       l_role_name VARCHAR2(100);
     BEGIN
       SELECT role INTO l_role_name FROM session_roles WHERE rownum = 1;
       dbms_output.put_line(CHR(10) || l_role_name);
     END;
     /
Procedure created.

ORAUSER1@DB> EXEC show_session_roles_invoker

CONNECT
PL/SQL procedure successfully completed.
```

Roles and DDLs

- Depending on the DDL statement, one or more privileges are needed to succeed, e.g.
  - To create a view on a table belonging to another user
    - **`CREATE VIEW`** or **`CREATE ANY VIEW`**
    - **`SELECT`** on this table or **`SELECT ANY TABLE`**
  - But these **`SELECT`** privileges cannot be granted through a role!
    - Views are definer's rights objects
- In general, when received through a role
  - All system and object privileges that permit a user to perform a DDL operation are usable, e.g.
    - System: **`CREATE TABLE, CREATE VIEW`**
    - Object: **`ALTER, INDEX`**
  - All system and object privileges that allow a user to perform a DML operation that is required to issue a DDL statement are not usable

Useful role-related views

View name	Describes
USER_ROLE_PRIVS	Roles directly granted to the current user
ROLE_ROLE_PRIVS	Roles granted to other roles (only roles to which the current user has access are listed)
ROLE_SYS_PRIVS	System privileges granted to roles (only roles to which the current user has access are listed)
ROLE_TAB_PRIVS	Object privileges granted to roles (only roles to which the current user has access are listed)
SESSION_ROLES	All enabled roles for the current user (except PUBLIC)

Thinking about security

- Never share your passwords
  - If access is required, separate account with the least privileges needed should be created
  - Responsibility easy to track with account management
- Separation of duties using database accounts
  - Reader
  - Writer
  - Owner

Increasing application security

- Using views
  - Privileges needed only for view, not its underlying objects
    - Security domain used when view is queried is of its definer (owner)
  - Can provide access to selected columns of base tables
  - Can provide access to selected rows (value-based security)
- Using stored procedures to encapsulate business logic
  - Privilege to update specific object only through procedure
  - Possibility to add more constraints, e.g.
    - Updates allowed only during business hours

Outline

- Users
- Privileges
- Roles
- Encryption
- SQL injection

Encryption (1/2)

- Way to increase protection for sensitive data
- Encryption using PL/SQL
  - **`DBMS_CRYPTO`** (replacing **`DBMS_OBFUSCATION_TOOLKIT`**)
- Transparent Data Encryption (TDE)
  - Oracle Enterprise Edition with Advanced Security option required
  - No application changes needed
  - Encryption of data before it’s written to storage
  - Decryption of data when it’s read from storage
  - Two modes supported
    - Tablespace encryption (11g) – hardware acceleration possible
    - Column encryption (10gR2)

Encryption (2/2)

- Additional protection for data in transit
  - Network encryption to protect communication to and from the database
  - Rejecting connections from clients without encryption
- Additional protection for backups
  - TDE encrypted data remains encrypted
  - Entire backup and export dump files encryption possibility

Outline

- Users
- Privileges
- Roles
- Encryption
- SQL injection

SQL injection defined

- Kind of attack with adding and executing unintended code from untrusted source
  - Manipulate select statements
  - Run DML or even DDL
  - Run stored procedures
- Virtually anything could be done in context of connected user privileges
  - Even more with definer’s right procedures
- Caused by
  - Wrong input handling – not only strings!
  - Implicit types conversions – dangerous

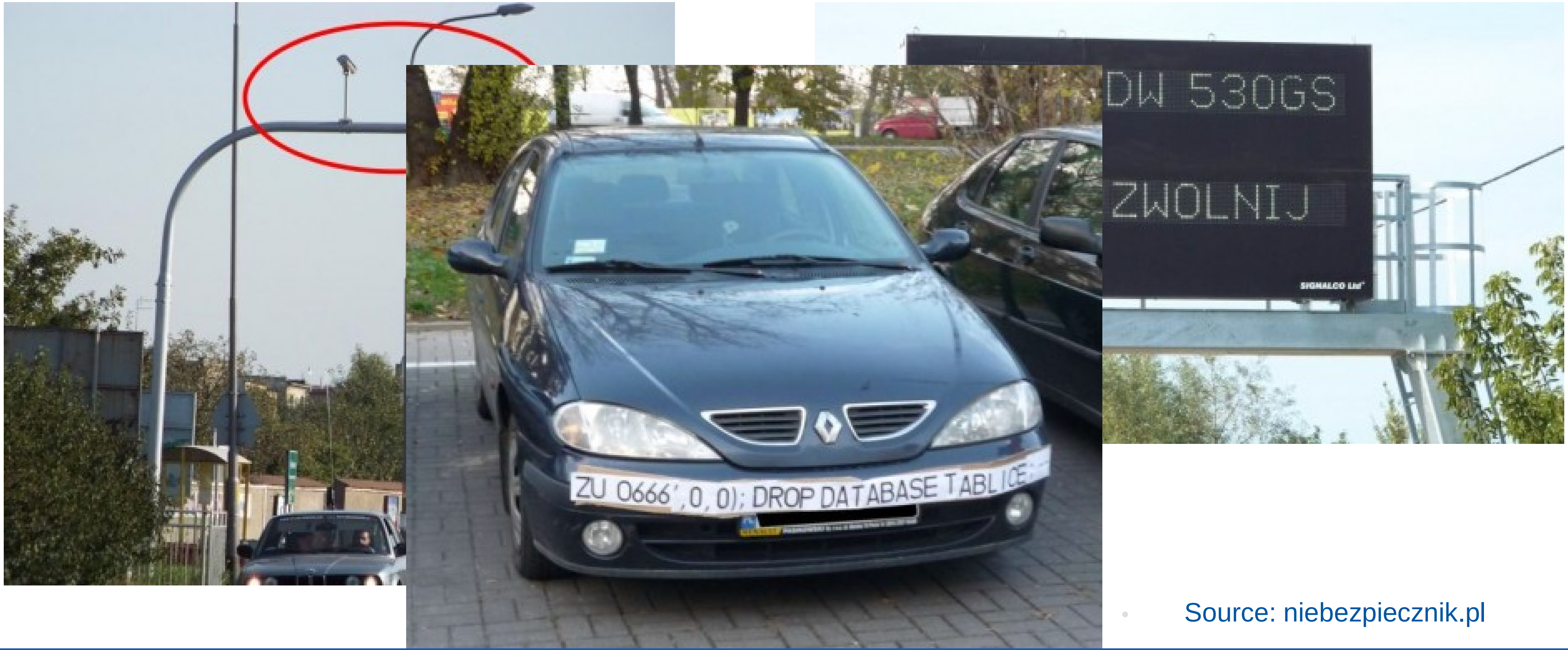
SQL injection prevention (1/2)

- Design security into your application from day 1
  - Detection very hard and time consuming in post-development phase
  - Could procedure without any input parameters be injected? Yes...
- Use bind variables!
  - You’ll be secure...
  - ...and will get better performance and scalability
- If not...
  - „then you must submit your code for review to at least five people who do not like you - they must be motivated to rip your code apart, critically review it, make fun of it - so they find the bugs” - Tom Kyte

SQL injection prevention (2/2)

- If you really have very good technical reasons not to use binds
  - Are you sure?
  - Use **`DBMS_ASSERT`** package to sanitize user inputs
  - Are you 100% sure?
- Don’t use implicit types conversions...
- ...and don’t rely on defaults
  - Application logic unintended change besides SQL injections

SQL injection – be prepared!



Source: niebezpiecznik.pl

SQL injection with inputs (1/4)

```
SQL> CREATE TABLE users (
       login VARCHAR2(20),
       pass VARCHAR2(20)
     );
Table created.

SQL> INSERT INTO users VALUES ('admin','pass');
1 row created.

SQL> COMMIT;
Commit complete.
```

SQL injection with inputs (2/4)

```
SQL> SELECT 1 allow FROM users
     WHERE login = 'admin' AND pass = 'fake';

no rows selected

SQL> SELECT 1 allow FROM users
     WHERE login = 'admin' AND pass = 'pass';

         ALLOW
--------------
             1
```

SQL injection with inputs (3/4)

```
SQL> SELECT 1 allow FROM users
     WHERE login = '&usr' AND pass = '&pwd';
Enter value for usr: admin
Enter value for pwd: fake' or 'a'='a
old   1: SELECT 1 allow FROM users WHERE login = '&usr' AND pass
         = '&pwd'
new   1: SELECT 1 allow FROM users WHERE login = 'admin'
         AND pass = 'fake' or 'a'='a'

         ALLOW
-------------
             1
```

SQL injection with inputs (4/4)

```
SQL> VARIABLE usr VARCHAR2(20);
SQL> VARIABLE pwd VARCHAR2(20);
SQL> EXEC :usr := 'admin';
PL/SQL procedure successfully completed.
SQL> EXEC :pwd := 'fake'' or ''a'' = ''a';
PL/SQL procedure successfully completed.
SQL> PRINT pwd
PWD
--------------------------------
fake' or 'a' = 'a

SQL> SELECT 1 allow FROM users WHERE login = :usr AND pass = :pwd;
no rows selected
```

SQL injection with inputs (1/7)

```
SQL> CREATE OR REPLACE PROCEDURE add_user (p_login VARCHAR2, p_pass
     VARCHAR2) AS
       l_cmd VARCHAR2(1000);
     BEGIN
       l_cmd := 'BEGIN
                 INSERT INTO users VALUES (''' || p_login || ''',''' ||
                                                  p_pass || ''');
                 COMMIT;
               END;';
       dbms_output.put_line(l_cmd);
       EXECUTE IMMEDIATE l_cmd;
     END;
     /
Procedure created.
```

SQL injection with inputs (2/7)

```
SQL> SET SERVEROUTPUT ON
SQL> SELECT * FROM users;
LOGIN                PASS
-------------------- --------------------
admin                pass

SQL> EXEC add_user('NewLogin','NewPass');
  BEGIN
    INSERT INTO users VALUES ('NewLogin','NewPass');
    COMMIT;
  END;
PL/SQL procedure successfully completed.
```

SQL injection with inputs (3/7)

```
SQL> SELECT * FROM users;

LOGIN                PASS
-------------------- --------------------
admin                pass
NewLogin             NewPass
```

SQL injection with inputs (4/7)

```
SQL> EXEC add_user('NewerLogin','NewerPass''); INSERT
     INTO users VALUES (''FakeUser'', ''FakePass'');--');

BEGIN
  INSERT INTO users VALUES ('NewerLogin', 'NewerPass');
  INSERT INTO users VALUES ('FakeUser', 'FakePass');--');
  COMMIT;
END;

PL/SQL procedure successfully completed.
```

SQL injection with inputs (5/7)

```
SQL> SELECT * FROM users;

LOGIN                PASS
-------------------- --------------------
NewerLogin           NewerPass
admin                pass
NewLogin             NewPass
FakeUser             FakePass
```

SQL injection with inputs (6/7)

```
SQL> EXEC add_user('NewestLogin','NewestPass'');
     EXECUTE IMMEDIATE ''DROP TABLE users'';--');

BEGIN
  INSERT INTO users VALUES ('NewestLogin','NewestPass');
  EXECUTE IMMEDIATE 'DROP TABLE users';--');
  COMMIT;
END;

PL/SQL procedure successfully completed.
```

SQL injection with inputs (7/7)

```
SQL> SELECT * FROM users;

SELECT * FROM users
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

SQL injection without inputs (1/10)

```
SQL> CREATE TABLE users (
       login  VARCHAR2(30),
       pass   VARCHAR2(30),
       expire TIMESTAMP
     );

Table created.


SQL> ALTER SESSION SET nls_timestamp_format = 'DD-MM-YYYY
     HH24:MI:SS';


Session altered.
```

SQL injection without inputs (2/10)

```
SQL> INSERT INTO users VALUES ('UserExpired', 'pass1234',
     localtimestamp - 1);

1 row created.

SQL> INSERT INTO users VALUES ('UserNotExpired', '4567pass',
     localtimestamp + 1);

1 row created.

SQL> COMMIT;
Commit complete.
```

SQL injection without inputs (3/10)

```
SQL> SELECT * FROM users;

LOGIN          PASS           EXPIRE
-------------- -------------- -------------------
UserExpired    pass1234       28-04-2013 11:47:28
UserNotExpired 4567pass       30-04-2013 11:47:32
```

SQL injection without inputs (4/10)

```
SQL> CREATE OR REPLACE PROCEDURE list_expired_users AS
       l_query       VARCHAR2(300);
       l_query_bind  VARCHAR2(300);
       l_time        TIMESTAMP;
       l_cur         SYS_REFCURSOR;
       l_login       VARCHAR2(30);
     BEGIN
       l_time := localtimestamp;
       l_query := 'SELECT login FROM users WHERE expire <=
                  ''' || l_time || '''';
       l_query_bind := 'SELECT login FROM users WHERE expire <=
                       :b_var';
```

SQL injection without inputs (5/10)

```
dbms_output.put_line('Concatenated query with implicit
                      conversions: ' || l_query);
OPEN l_cur FOR l_query;
LOOP
  FETCH l_cur INTO l_login;
  EXIT WHEN l_cur%NOTFOUND;
  dbms_output.put_line(l_login);
END LOOP;
CLOSE l_cur;
```

SQL injection without inputs (6/10)

```
    dbms_output.put_line('Bind variable query: ' ||
                         l_query_bind);
    OPEN l_cur FOR l_query_bind USING l_time;
    LOOP
      FETCH l_cur INTO l_login;
      EXIT WHEN l_cur%NOTFOUND;
      dbms_output.put_line(l_login);
    END LOOP;
    CLOSE l_cur;
  END;
  /
```

SQL injection without inputs (7/10)

```
SQL> SELECT value FROM v$nls_parameters
     WHERE parameter = 'NLS_TIMESTAMP_FORMAT';

VALUE
----------------------------------------
DD-MM-YYYY HH24:MI:SS

SQL> SET SERVEROUTPUT ON
```

SQL injection without inputs (8/10)

```
SQL> EXEC list_expired_users;
Concatenated query with implicit conversions: SELECT
login FROM users WHERE expire <= '28-04-2013 11:53:21'
UserExpired

Bind variable query: SELECT login FROM users WHERE expire
<= :b_var
UserExpired

PL/SQL procedure successfully completed.
```

SQL injection without inputs (9/10)

```
SQL> ALTER SESSION SET nls_timestamp_format = '"'' UNION
     SELECT login || '' '' || pass FROM users--"';
Session altered.
SQL> SELECT value FROM v$nls_parameters
     WHERE parameter = 'NLS_TIMESTAMP_FORMAT';
VALUE
------------------------------------------------------
"' UNION SELECT login || ' ' || pass FROM users--"
SQL> SELECT localtimestamp FROM dual;
LOCALTIMESTAMP
------------------------------------------------------
' UNION SELECT login || ' ' || pass FROM users--
```

SQL injection without inputs (10/10)

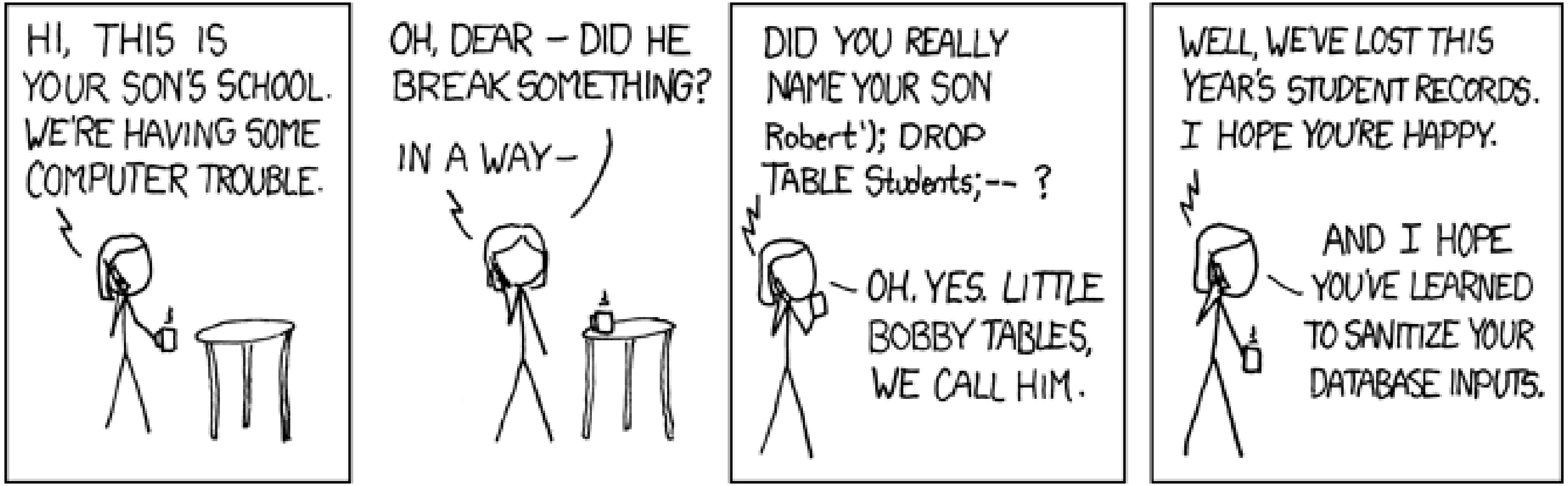
```
SQL> EXEC list_expired_users;

Concatenated query with implicit conversions:
SELECT login FROM users WHERE expire <= '' UNION SELECT login ||
' ' || pass FROM users--'
UserExpired pass1234
UserNotExpired 4567pass

Bind variable query:
SELECT login FROM users WHERE expire <= :b_var
UserExpired
PL/SQL procedure successfully completed.
```

Questions?

Thank you!


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

www.cern.ch