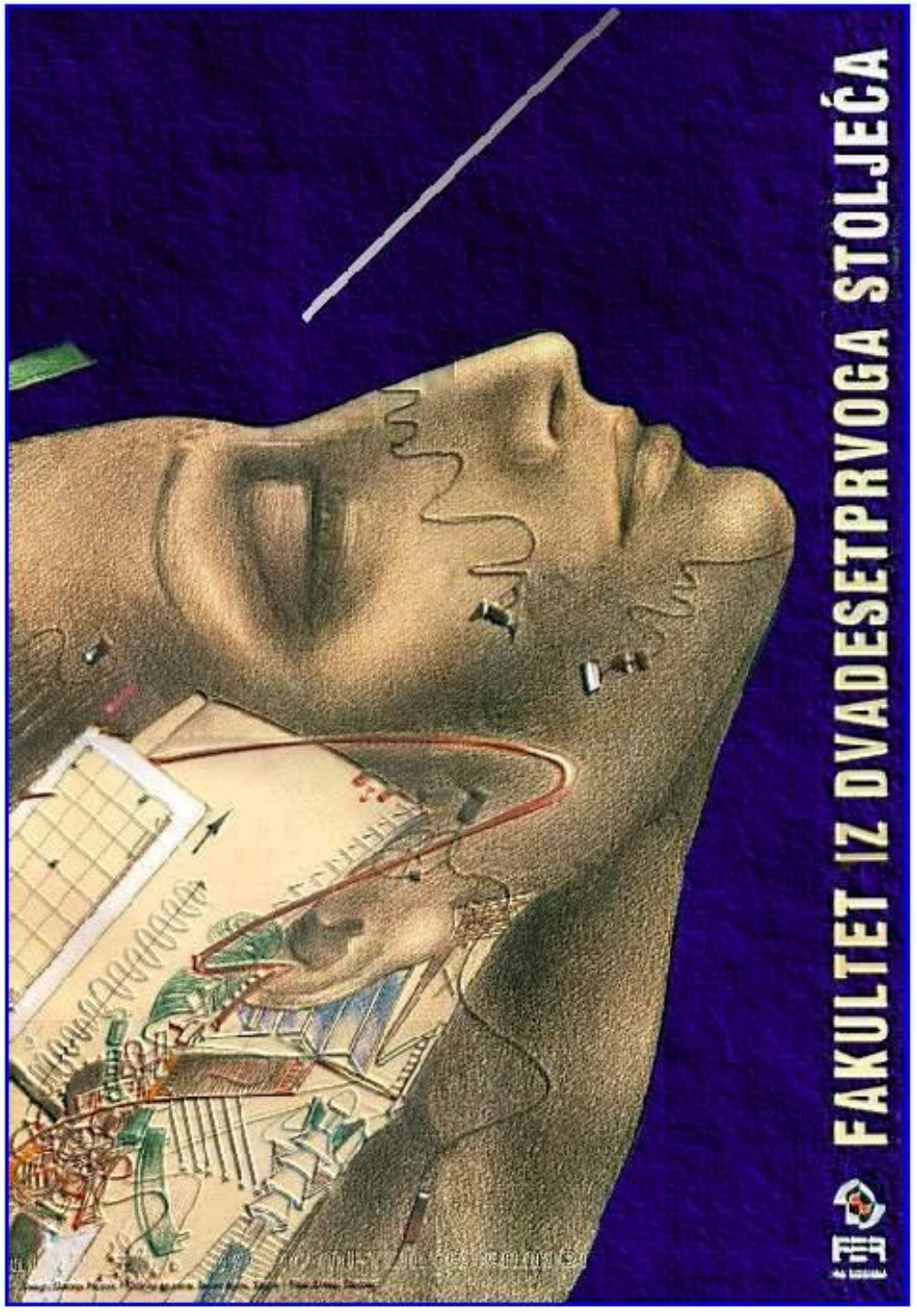
Databases

Lectures

June 2014.

19. Database security



Database integrity and security

- Database integrity and security are terms often discussed together, but they are two separate aspects of data protection
  - Database integrity – ensures that operations over data performed by users **are correct** (i.e. they always result in a consistent database state)
    - "the protection of data against authorized access"
  - Database security – ensures that users performing operations on data **are authorized** to perform those operations
    - "the protection of data against unauthorized access"

There are similarities between these two terms. In both cases:

- **rules** that users should not violate must be defined
- rules are stored within the data dictionary
- DBMS monitors user activity - ensures compliance with the rules

Types of security violations and possible consequences

- Database security violations:
  - unauthorized reading of data
  - unauthorized modification of data
  - unauthorized destruction of data
- Possible consequences:
  - theft or fraud
  - loss of confidentiality
    - disclosure of information critical to the functioning of the company
    - e.g. stealing of formulas - resulting in the loss of competitiveness in the market
  - loss of privacy
    - disclosure of personal data
    - e.g. stealing of data about person's health - resulting with a court process against the database owner
  - loss of availability
    - e.g. caused by destruction of data

Security countermeasures

- to protect the database, security measures at several levels must be taken:
  - **database system level**
    - prevent access to databases or database objects for which users are not authorized
  - **operating system level**
    - prevent access to main memory segments or files where the DBMS stores data
  - **network level**
    - prevent internet and intranet traffic interception (*sniffing*)
  - **physical level**
    - physically protect the location of the computer system
  - **human level**
    - prevent authorized users to unintentionally or intentionally (e.g. in exchange for a bribe or other personal benefit) provide access to information to unauthorized persons

Aspects of data protection

- **legal, social and ethical aspect**
  - does the database owner have a legal right to collect and use data
  - e.g. should a health institution which collects data about patients (in accordance with the law), use the same information in making a decision whether to hire one of its patients
- **strategic aspect**
  - who defines access rules - who determines what privileges a particular database user has, ...
- **operational aspect**
  - how to ensure compliance with the rules - using which mechanisms is compliance with defined rules ensured by, in what way are passwords protected, how often passwords have to be changed, ...

The Constitution of the Republic of Croatia - Article 37

Everyone shall be guaranteed the safety and secrecy of personal data. Without consent from the person concerned, personal data may be collected, processed and used only under conditions specified by law.

Protection of data and supervision of the work of information systems in the Republic shall be regulated by law.

The use of personal data contrary to the purpose of their collection shall be prohibited.

- Act on Personal Data Protection

DBMS users and authentication

- system administrator (operating system or DBMS administrator) allows a user access to the system (operating system or DBMS) by defining a unique user identifier (*user name*, *user ID*, *login ID*) and associated *password* known only to the respective user and the system
- user accessing the system (operating system or DBMS) must prove his authenticity by typing a password (*authentication*)
- when performing user authentication, DBMS can use:
  - operating system mechanisms

    or
  - proprietary mechanisms

Authorization and access control models

- Authorization is the process of assigning permissions for performing specific operation types (read, change, delete, ...) over specific database objects (relation, view, attribute, ...) to a specific user.
  - data about granted permissions are stored within data dictionary
- Before performing any operation, DBMS checks whether the user has the permission to perform that operation on the object.
  - access control
- DBMSs support two different access control models:
  - **Mandatory Access Control** (MAC)
  - **Discretionary Access Control** (DAC)

Mandatory Access Control

- supported by a smaller number of DBMSs
  - used relatively rarely compared to discretionary access control
- applicable to systems where privileges are assigned based on a user's position in the organization hierarchy (military, government, ...)
- each **object** is assigned a security class (*classification level*), e.g. confidential, secret, top secret, ...
- each **subject** is assigned a clearance for a security class (*clearance level*)
  - users can perform operations on those objects for which they have the appropriate clearance level

Discretionary Access Control

- discretionary access control is supported by most of today's DBMSs
  - supported by the SQL standard
- a particular user is explicitly granted permission to perform a particular operation on a particular object
  - permissions are described with triplets <user, object, operation type>

    <horvat, exam, read>

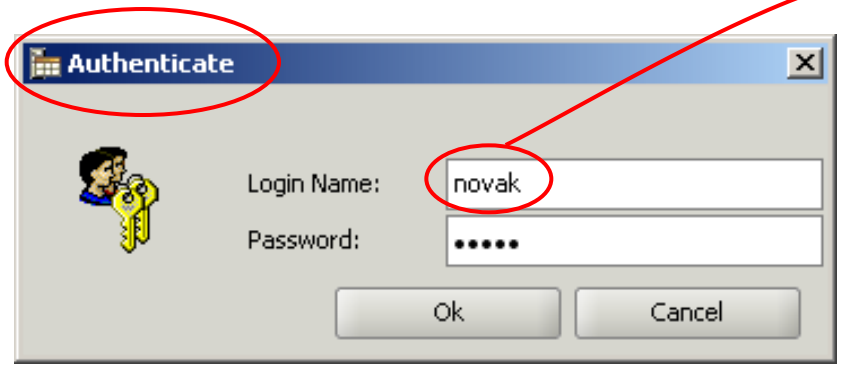
    <horvat, exam, change>

    <horvat, course, read>

    <novak, course, read>
  - when the user novak attempts to perform a read operation on the object (relation) course, DBMS checks if the permission in the form of triplet <novak, course, read> exists
- in the rest of this lecture, discretionary access control will be considered further

SQL users

- user with a specific user identifier (*userID*)
  - when establishing an SQL-session, user logs in using his user identifier and then verifies authenticity by typing in the password
  - the function USER returns the user identifier that is used in a given SQL-session




```
SELECT FIRST 1 USER AS user_name
  FROM mjesto;
```

user_name
novak

- any user (PUBLIC)
  - by assigning a permission to the "user" PUBLIC, the permission is granted to all current and future users

SQL objects and object owners

- **Objects**
  - relation (*table*)
  - attribute (*column*)
  - virtual relation (*view*)
  - database
- **Object owner**
  - the owner is the user who created the object, e.g.:
    - database owner is the user who created the database
    - relation owner is the user who created the relation
  - the owner of an object implicitly has **all privileges** on that object, including permission to:
    - grant any permissions on that object to other users
    - delete that object

Database-level privileges (*dbPrivilege*)

- Different DBMSs have different ways for assigning privileges at the database level. IBM Informix DBMS uses following permissions:
  - **CONNECT**
    - user can establish an SQL-session, perform operations on objects for which he already has permissions (user has obtained permission from the object owner or is the object owner) and can create virtual and temporary relations
  - **RESOURCE**
    - CONNECT + can create **new** relations in the database
  - **DBA**
    - RESOURCE + regardless of the ownership and permissions on database objects: perform any operation on any database object, delete any database object (including the entire database)
    - user who created the database is the database owner and implicitly gets DBA (Database Administrator) permission

[Virtual] Relation-level privileges (*tablePrivilege*)

- **SELECT [(*columnList*)]**
  - read tuples (or values of specified attributes) from a [virtual] relation
- **UPDATE [(*columnList*)]**
  - modify tuples (or values of specified attributes) in a [virtual] relation
- **INSERT**
  - insert tuples in a [virtual] relation
- **DELETE**
  - delete tuples from a [virtual] relation
- **REFERENCES [(*columnList*)]**
  - the right to refer to the relation (or to specified attributes only) as a referenced relation in a foreign key definition
- **INDEX**
  - create indexes for a **relation**
- **ALTER**
  - change the structure of a **relation** and define integrity constraints
- **ALL PRIVILEGES**
  - all relation-level privileges listed above

SQL statements for granting and revoking privileges

- GRANT *dbPrivilege* TO { PUBLIC | *userList* }
- REVOKE *dbPrivilege* FROM { PUBLIC | *userList* }
- GRANT *tablePrivilegeList* ON { *tableName* | *viewName* }
  TO { PUBLIC | *userList* | *roleList* }
  [WITH GRANT OPTION]
- REVOKE *tablePrivilegeList* ON { *tableName* | *viewName* }
  FROM { PUBLIC | *userList* | *roleList* }
  [CASCADE | RESTRICT]

Example 1:

student

studId	FName	LName	zip	address
100	Ana	Ivić	51000	Korzo 2
102	Ivan	Perić	10000	Ilica 20
105	Matija	Matić	31000	Unska 7
107	Tea	Bilić	10000	Vlaška 5

exam

studId	courseName	dateOfExam	grade
100	Physics	1.5.2010	3
102	Mathematics	7.9.2009	1
102	Mathematics	9.2.2010	5
107	Physics	5.4.2012	4

- create the database studDB and the relations student and exam
  - database and relations owner is the user bpadmin
- user horvat must have permissions to:
  - read all data in the relations student and exam
  - insert, modify and delete of all data from the relation exam
- user novak must have permissions to:
  - read all data from the relation student
  - modify the value of the postal code and the address in the relation student
- user kolar must have the permission to:
  - read all data from the relation student, except address

Example 1 (cont.):

bpadmin ← **user bpadmin executes following statements**

```
CREATE DATABASE studDB;
CREATE TABLE student (...);
CREATE TABLE exam (...);

GRANT CONNECT TO horvat;
GRANT CONNECT TO novak;
GRANT CONNECT TO kolar;

GRANT SELECT ON student
  TO horvat;
GRANT SELECT, INSERT
    , UPDATE, DELETE ON exam
  TO horvat;

GRANT SELECT ON student
  TO novak;
GRANT UPDATE(zip, address)
  ON student TO novak;

GRANT SELECT(studId, FName
            , LName, zip)
  ON student TO kolar;
```

- user bpadmin is an owner of the database studDB, as well as relations student and exam. He owns the DBA database-level privilege.
- privilege for establishing an SQL-session
- user horvat has the permission to read data from the relation student
- user horvat has permissions to read, modify and delete data from the relation exam
- user novak has the permission to read data from the relation student
- user novak has the permission to modify values of attributes zip and address from the relation student
- user kolar has the permission to read values of all attributes in the relation student, except the attribute address

Example 2:

bpadmin

```
CREATE DATABASE studDB;
GRANT RESOURCE TO horvat;
GRANT CONNECT TO novak;
```

user bpadmin creates the database studDB. As the database owner, he implicitly gets the DBA database-level privilege.

horvat

```
CREATE TABLE county (
  countyId   INTEGER
, countyName CHAR(30)
, PRIMARY KEY(countyId));
GRANT SELECT, INSERT, UPDATE
  ON county TO novak;
```

can be done, because horvat has the RESOURCE permission

can be done, because horvat is the owner of the relation county

novak

```
SELECT * FROM county;
INSERT INTO county ...;
UPDATE county ...;
```

can be done, because novak has the CONNECT permission (SQL-session without at least CONNECT permission could not be established), as well as permissions received from the owner of the relation county

Example 2 (cont.):

novak

```
DROP TABLE county;
```

can not be done, because novak is not the owner of the relation and does not have the DBA permission

kolar

```
SELECT * FROM county;
```

can not be done, because kolar does not have at least the CONNECT permission (SQL-session could not be established)

horvat

```
GRANT CONNECT TO kolar;
```

can not be done, because horvat does not have the DBA permission

bpadmin

```
GRANT CONNECT TO kolar;
```

can be done, because bpadmin has the DBA permission

horvat

```
GRANT SELECT
  ON county TO kolar;
```

can be done, because horvat owns the relation county

kolar

```
SELECT * FROM county;
```

can be done, because kolar has (at least) the CONNECT permission, as well as the read permission for the relation county

Example 2 (cont.):

novak

```
CREATE TABLE town ...;
```

can not be done, because novak does not have the RESOURCE permission

horvat

```
GRANT RESOURCE TO novak;
```

can not be done, because horvat does not have the DBA permission

bpadmin

```
GRANT DBA TO horvat;
```

can be done, because bpadmin has the DBA permission

horvat

```
GRANT RESOURCE TO novak;
```

can be done, because horvat has the DBA permission

novak

```
CREATE TABLE town (...
  REFERENCES county ...);
```

can not be done, because novak does not have the permission for creating the foreign key that references the primary key of the relation county (he can create the relation without the foreign key because he has the RESOURCE permission)

Example 2 (cont.):

horvat

```
GRANT REFERENCES
  ON county TO novak;
```

can be done, because horvat has the DBA permission (it can be done even withouth the DBA permission, because he is the owner of the relation county)

novak

```
CREATE TABLE town (...
  REFERENCES county ...);
```

can be done, because novak has the RESOURCE permission and the permission for creating the foreign key that references the primary key of the relation county

horvat

```
GRANT CONNECT TO PUBLIC;
```

can be done, because horvat has the DBA permission

- every (present or future) user who successfully verifies his authenticity can now establish an SQL-session with the database studDB

novak

```
GRANT SELECT
  ON town TO PUBLIC;
```

can be done, because novak is the owner of the relation town

- every (present or future) user who successfully establishes an SQL-session with the database (and verifies his authenticity) can now execute a SELECT statement on the relation town

Propagation of privileges

- if a privilege is granted to a user with option WITH GRANT OPTION, the user can also grant the privilege to other users (even though he is not an object owner)

Example:

user1

```
CREATE TABLE exam (...);
GRANT SELECT ON exam TO user2 WITH GRANT OPTION;
GRANT SELECT ON exam TO user3 WITH GRANT OPTION;
```

user2

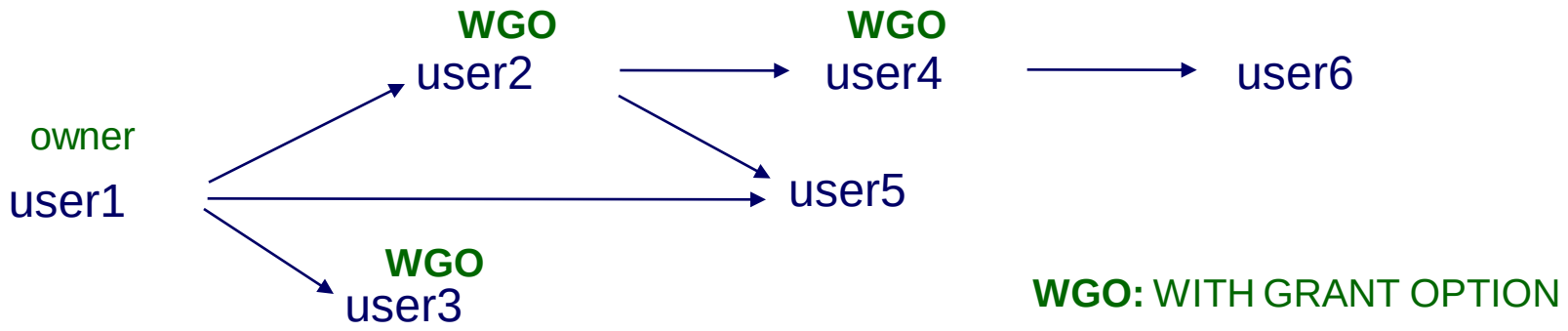
```
GRANT SELECT ON exam TO user4 WITH GRANT OPTION;
GRANT SELECT ON exam TO user5;
```

user4

```
GRANT SELECT ON exam TO user6;
```

user1

```
GRANT SELECT ON exam TO user5;
```



Revoking privileges

- user can revoke a privilege he granted, using the REVOKE statement

Example:
- user bpadmin is the owner of the database studDB
- user horvat is the owner of the relation town

horvat

```
GRANT SELECT, UPDATE ON town TO novak WITH GRANT OPTION;
```

novak

```
GRANT SELECT, UPDATE ON town TO kolar;
```

- e.g. statement:

```
REVOKE UPDATE ON town FROM kolar;
```

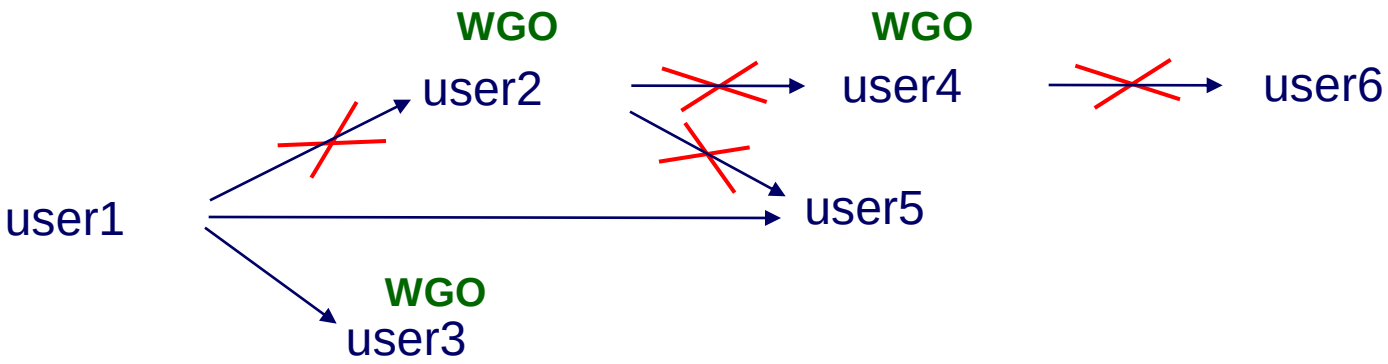
  can be peformed by user novak, because novak is the user that granted this privilege

Revoking privileges granted based on WITH GRANT OPTION

- when revoking a privilege from a user x (who propagated the privilege based on option WITH GRANT OPTION) using the option CASCADE, the privilege will also be revoked from all other users who gained the privilege from the user x (directly or indirectly)

Example: `user1`

```
REVOKE SELECT ON exam FROM user2 CASCADE;
```

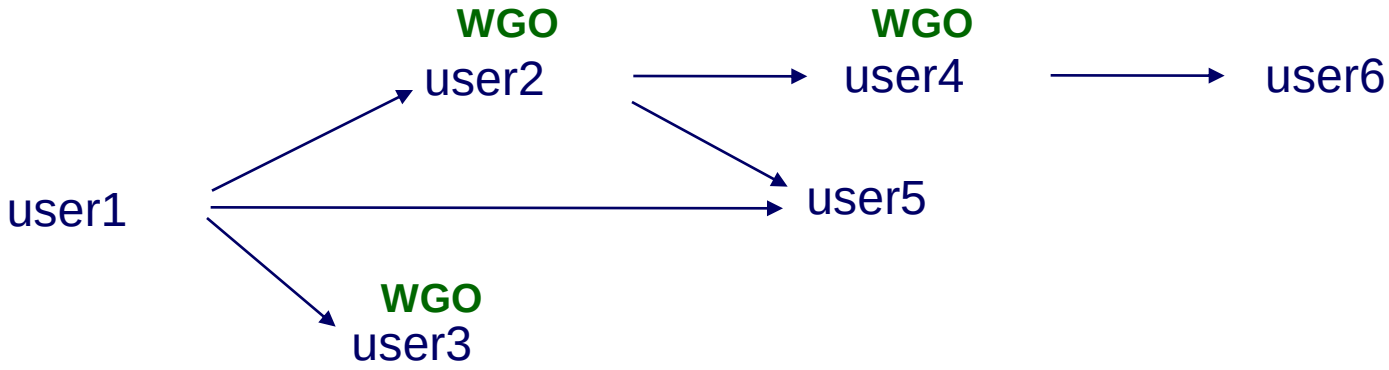


- after executing the statement, privilege is removed from users user2, user4 and user6
- user5 will lose the permission gained from user2, but will retain the permission gained from user1
- option CASCADE is the default option

Ukidanje dozvola dodijeljenih temeljem WITH GRANT OPTION

- when revoking a privilege from the user x, using the option RESTRICT, the privilege will be revoked only if the user x did not propagate the privilege based on option WITH GRANT OPTION

Example:



user1

```
REVOKE SELECT ON exam FROM user2 RESTRICT;
```

DBMS refuses to execute the statement (returns an error)

user2

```
REVOKE SELECT ON exam FROM user4 RESTRICT;
```

DBMS refuses to execute the statement (returns an error)

user1

```
REVOKE SELECT ON exam FROM user3 RESTRICT;
```

DBMS executes the statement (revokes the privilege from the user3)

Using virtual relations

exam

studId	courseName	dateOfExam	grade
100	Physics	1.5.2010	3
102	Mathematics	7.9.2009	1
102	Mathematics	9.2.2010	5
107	Physics	5.4.2012	4

- user horvat is the owner of the relation exam
- allow to the user novak to read only average grade for each course
- allow to the user kolar to read, insert, modify and delete data only for the Physics exams

horvat

```
CREATE VIEW averageC (courseName, avgGrade) AS
  SELECT courseName, AVG(grade)
    FROM exam
    GROUP BY courseName;
GRANT SELECT ON averageC TO novak;

CREATE VIEW examPhysics AS
  SELECT * FROM exam
    WHERE courseName = 'Physics'
  WITH CHECK OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE
  ON examPhysics TO kolar;
```

- it is necessary to create a virtual relation examPhysics with the option WITH CHECK OPTION! Why?

Granting context-dependent privileges

exam

studId	courseId	dateOfExam	grade
100	100	1.5.2010	3
102	200	7.9.2009	1
102	200	9.2.2010	5
107	300	5.4.2012	4

teacher

teachId	teachFName	teachLName	userId
1001	Slavko	Kolar	kolar
1002	Ivo	Ban	ban
1003	Ana	Novak	novak

teaches

teachId	courseId
1001	100
1001	200
1002	200
1003	200
1003	300

- user horvat is the owner of all relations
- every teacher (users kolar, ban, novak) must be able to read and change exams only for courses they teach

BAD SOLUTION!

horvat

```
CREATE VIEW kolarExams AS
  SELECT * FROM exam
    WHERE courseId IN (
      SELECT courseId FROM teaches
        WHERE teachId = 1001) WITH CHECK OPTION;
GRANT SELECT, UPDATE ON kolarExams TO kolar;
```

- repeat for every teacher: banExams, novakExams, ...
- new virtual relation for every new teacher (≈150 on FER)
- every teacher must write queries over the relation exam in a different way

Granting context-dependent privileges

exam

studId	courseId	dateOfExam	grade
100	100	1.5.2010	3
102	200	7.9.2009	1
102	200	9.2.2010	5
107	300	5.4.2012	4

teacher

teachId	teachFName	teachLName	userId
1001	Slavko	Kolar	kolar
1002	Ivo	Ban	ban
1003	Ana	Novak	novak

teaches

teachId	courseId
1001	100
1001	200
1002	200
1003	200
1003	300

RIGHT SOLUTION!

horvat

```
CREATE VIEW examForTeacher  AS
  SELECT * FROM exam
    WHERE courseId IN (
      SELECT courseId FROM teaches, teacher
       WHERE teaches.teachId = teacher.teachId
         AND userId = USER) WITH CHECK OPTION;
GRANT SELECT, UPDATE ON examForTeacher  TO kolar;
GRANT SELECT, UPDATE ON examForTeacher  TO ban;
GRANT SELECT, UPDATE ON examForTeacher  TO novak;
```

- "content" of the virtual relation will depend on the user identifier of a teacher who established an SQL-session
- should teachers be permitted to change values of the attribute userId in the relation teacher, or to change content of the relation teaches?!

Using synonyms

PROBLEM:

- teachers (i.e. application programs that teachers use) must use the virtual relation examForTeacher in queries about exams :

```
SELECT * FROM examForTeacher WHERE grade = 1;
```

- dean (whose username is e.g. *novosel*), as opposed to teachers, gets all privileges on the relation exam. In queries about exams, he must use the relation exam:

```
SELECT * FROM exam WHERE grade = 1;
```

- when someone else becomes a dean, privileges on the relation exam will be revoked from the user *novosel*, and he will receive privileges on the virtual relation examForTeacher . He will then have to use the virtual relation examForTeacher in queries:

```
SELECT * FROM examForTeacher  WHERE grade = 1;
```

Using synonyms

SOLUTION:

- Creating synonyms: alternate names for [virtual] tables

user with DBA permission

```
CREATE PRIVATE SYNONYM kolar.examForAll FOR examForTeacher;
CREATE PRIVATE SYNONYM ban.examForAll FOR examForTeacher;
... synonyms for all other teachers and synonym for a dean
CREATE PRIVATE SYNONYM novosel.examForAll FOR exam;
```

- now the dean and teachers can use the same object name in their queries about exams

```
SELECT * FROM examForAll WHERE grade = 1;
```

- when the user novosel ceases to be a dean

horvat

```
REVOKE SELECT, UPDATE ON exam FROM novosel;
GRANT SELECT, UPDATE ON examForTeacher TO novosel;
```

user with DBA permission

```
DROP SYNONYM novosel.examForAll;
CREATE PRIVATE SYNONYM novosel.examForAll FOR examForTeacher;
```

- user novosel will continue to use the object name examForAll in his queries, but query results will contain only data for which he has permission as a teacher

Granting the same privileges to a large number of users

PROBLEM:

- every teacher should have privileges to read, insert and change information about exams for courses that he teaches, read data from relations teacher, teaches, etc.
  - 150 teachers ⇒ a series of statements granting permissions must be executed 150 times :

```
GRANT SELECT, INSERT, UPDATE ON examForTeacher TO kolar;
GRANT SELECT ON course TO kolar;
GRANT SELECT ON teacher TO kolar;
...
-- repeat for each of the 150 teachers
```

- repeat the process for every new employed teacher
- when the teacher retires, a series of REVOKE statements must be executed
- if access rules are changed (for example, if it was decided that teachers could delete "their" exams), changes must be performed for every teacher:

```
GRANT DELETE ON examForTeacher TO kolar;
-- repeat for each of the 150 teachers
```

Granting the same privileges to a large number of users

SOLUTION:

- a role is defined, e.g. role teacherR
- permissions are assigned to the role, instead of directly to each user-teacher:

```
CREATE ROLE teacherR;
GRANT SELECT, INSERT, UPDATE ON examForTeacher TO teacherR;
GRANT SELECT ON teacher TO teacherR;
GRANT SELECT ON teaches TO teacherR;
...
```

- instead of a list of permissions, it is enough to grant a single permission for using the role teacherR to each teacher:

```
GRANT teacherR TO kolar;
GRANT teacherR TO ban;
...
```

- if the teacher with the user name ban retires:

```
REVOKE teacherR FROM ban;
```

- if teachers need to get permission to delete "their" exam:

```
GRANT DELETE ON examsForTeach TO teacherR;
```

Using permissions obtained through roles

- after an SQL-session is established, a user has the following permissions:
  1. all permissions that are granted to the "user" PUBLIC
  2. all permissions that are granted directly to the respective user
  3. all permissions on objects for which the respective user is an owner
  4. database-level permissions (for example, if the user has DBA privilege, he is allowed to perform all operations on all objects)
- if the user intends to use permissions granted to a role, he must execute the statement SET ROLE, for example:

```
SET ROLE teacherR;
```

  - from that moment on, the user will have permissions granted to the role teacherR (in addition to permissions listed under 1-4)
- a user can be granted more than one role, but in the user session he can have only one of those roles active at any point in time. For example, after executing the statement:

```
SET ROLE studentAdvisor;
```

  - the user will have permissions granted to the role studentAdvisor (in addition to permissions listed under 1-4), but permissions granted only through the role teacherR will be disabled
- when the user doesn't want to use permissions obtained through any role, he can execute the statement:

```
SET ROLE NONE;
```

Database auditing

- recording all access to sensitive data in a special file (*Audit Trail*)
- a typical audit trail record contains the following information:
  - executed SQL statement (*statement source*)
  - location of the request (from which terminal, IP address)
  - user (user name) who performed the operation
  - date and time when the operation what executed
  - tuples, attributes to which the request applies
  - old tuple value
  - new tuple value
- the very fact that all performed operations on data are recorded is often sufficient to prevent misuse