Integrity and Security

CENG315 INFORMATION MANAGEMENT

Data Integrity and Security

- The data needs to have integrity and be secure.
- Integrity: the database should not become incorrect or inconsistent due to an inadvertent update (such as a typo or misunderstanding).
- Security: the data should not be accessible to unauthorized people (whether they be curious existing users, or malicious non-users).

Data Integrity and Security (Cont.)

- We consider three topics:
  - Assertions
  - Triggers
  - Authorization

Constraints

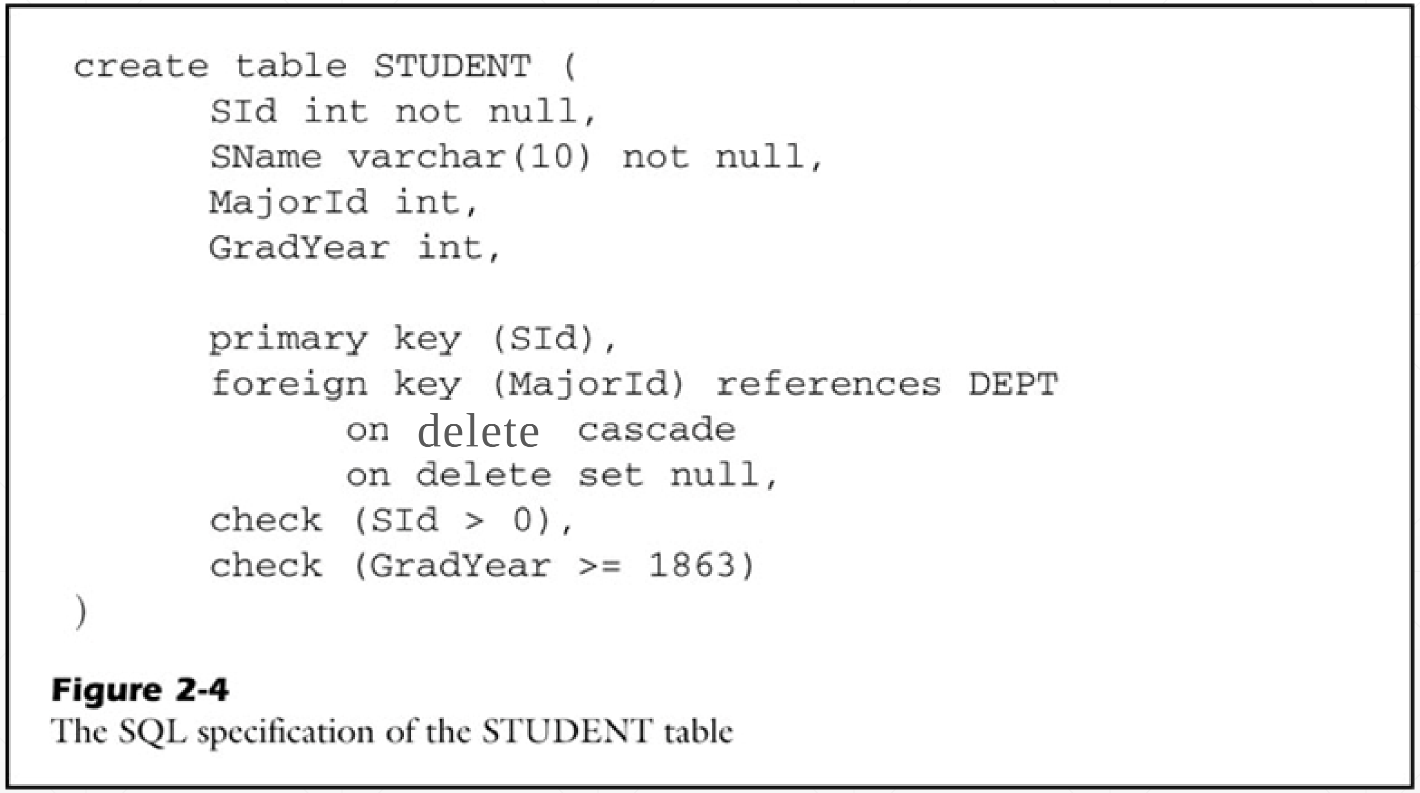
- A ***constraint*** describes the allowable states that the tables in the database may be in.
- Whenever a user submits an update statement, the database system first checks the requested changes against the constraints and performs the update if there is no violation.
- In this lecture we are interested in ***integrity constraints***.

Integrity Constraints

- An ***integrity constraint*** encodes “business rules” about the organization.
- Integrity constraints have two purposes:
  - They can detect bad data entry.
  - They can enforce the “rules” of the organization.
- An integrity constraint may apply to
  - an individual record, “a student’s graduation year is at least 1863”
  - a table, “a professor teaches at most two sections per year”
  - or database, “a student cannot take a course more than once”

Integrity Constraints (Cont.)

- We can specify an integrity constraint within an SQL ***create table*** statement.



```
create table STUDENT (
      SId int not null,
      SName varchar(10) not null,
      MajorId int,
      GradYear int,

      primary key (SId),
      foreign key (MajorId) references DEPT
            on delete  cascade
            on delete set null,
      check (SId > 0),
      check (GradYear >= 1863)
)
```

Figure 2-4
The SQL specification of the STUDENT table

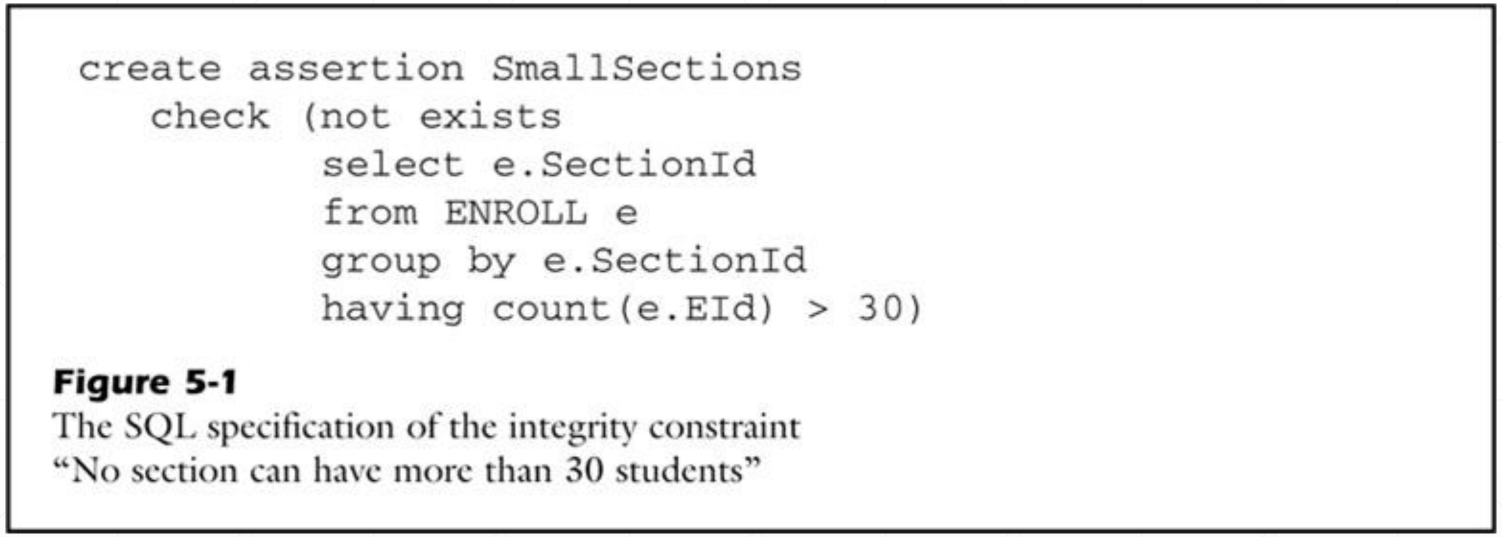
Integrity Constraints (Cont.)

- Can a ***create table*** statement specify an integrity constraint that applies to an entire table or to multiple tables?

Integrity Constraints (Cont.)

- Can a ***create table*** statement specify an integrity constraint that applies to an entire table or to multiple tables?
  - NO!
  - Can only specify a fraction of the possible integrity constraints, namely those constraints that apply to the individual records in a table.
  - We need to use assertions in order to specify an integrity constraint that applies to an entire table or to multiple tables.

Assertions

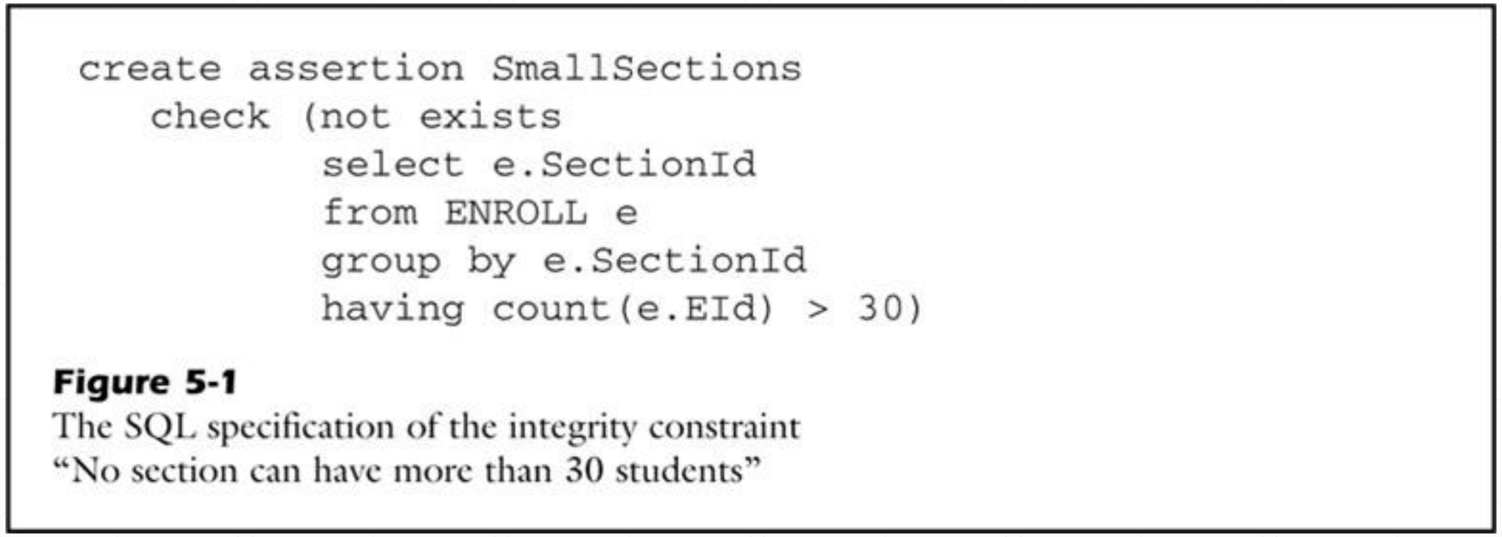


```
create assertion SmallSections
   check (not exists
          select e.SectionId
          from ENROLL e
          group by e.SectionId
          having count(e.EId) > 30)
```

Figure 5-1
The SQL specification of the integrity constraint "No section can have more than 30 students"

- An assertion is a predicate that must always be satisfied by the database.
- The ***check*** clause contains a predicate inside some parentheses.
  - check (predicate)

Assertions (Cont.)

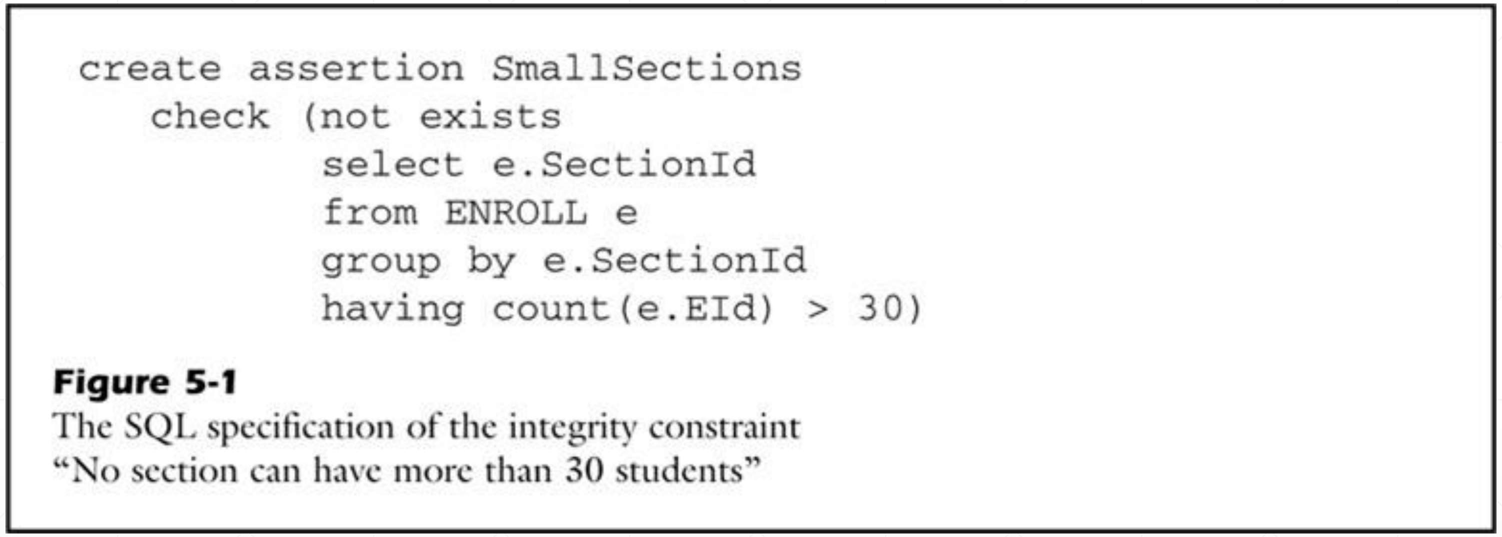


```
create assertion SmallSections
   check (not exists
          select e.SectionId
          from ENROLL e
          group by e.SectionId
          having count(e.EId) > 30)
```

Figure 5-1
The SQL specification of the integrity constraint
“No section can have more than 30 students”

- The predicate in an assertion is of the form “***not exists Q***” for a query *Q*.
  - The operator “*not exists*” returns *true* if the output table of its query contains no records.

Assertions (Cont.)



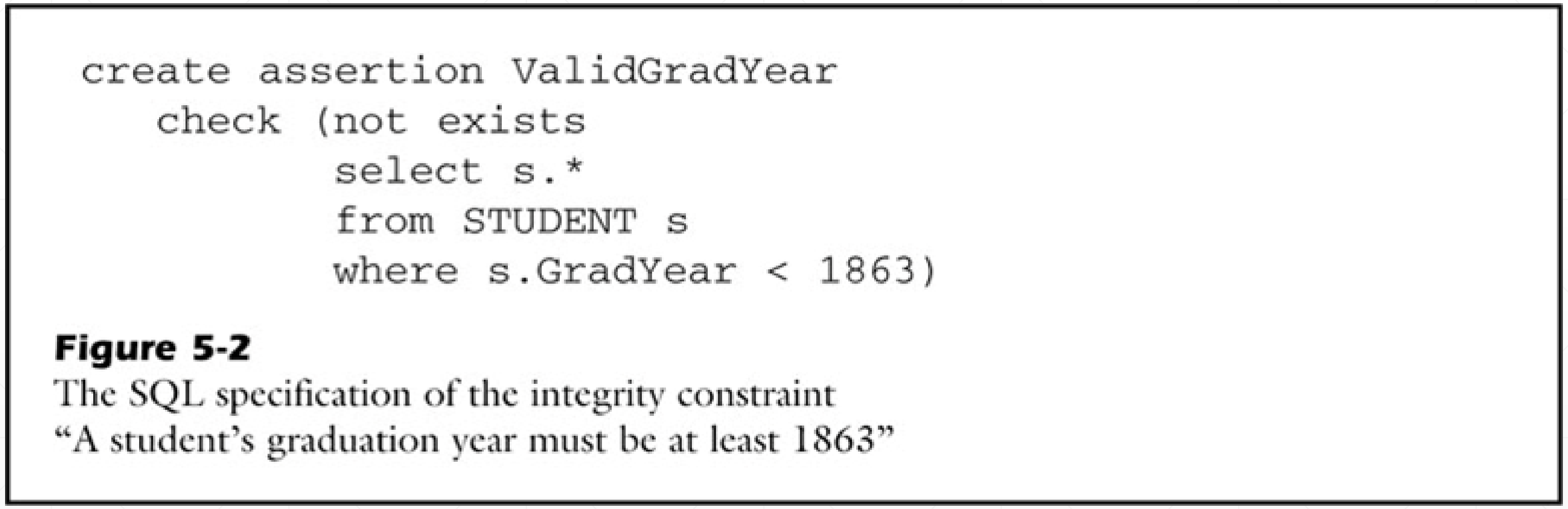
```
create assertion SmallSections
   check (not exists
          select e.SectionId
          from ENROLL e
          group by e.SectionId
          having count(e.EId) > 30)
```

Figure 5-1
The SQL specification of the integrity constraint
"No section can have more than 30 students"

- The predicate in an assertion can be arbitrary SQL predicate, but most commonly it is the form *"not exists Q*".
- The query *Q* finds violations of the constraint (such as existence of a large section), the predicate "*not exists Q*" then asserts that there are no such violations.

Assertions (Cont.)

- An assertion can denote any constraint whose violation can be expressed as a query.

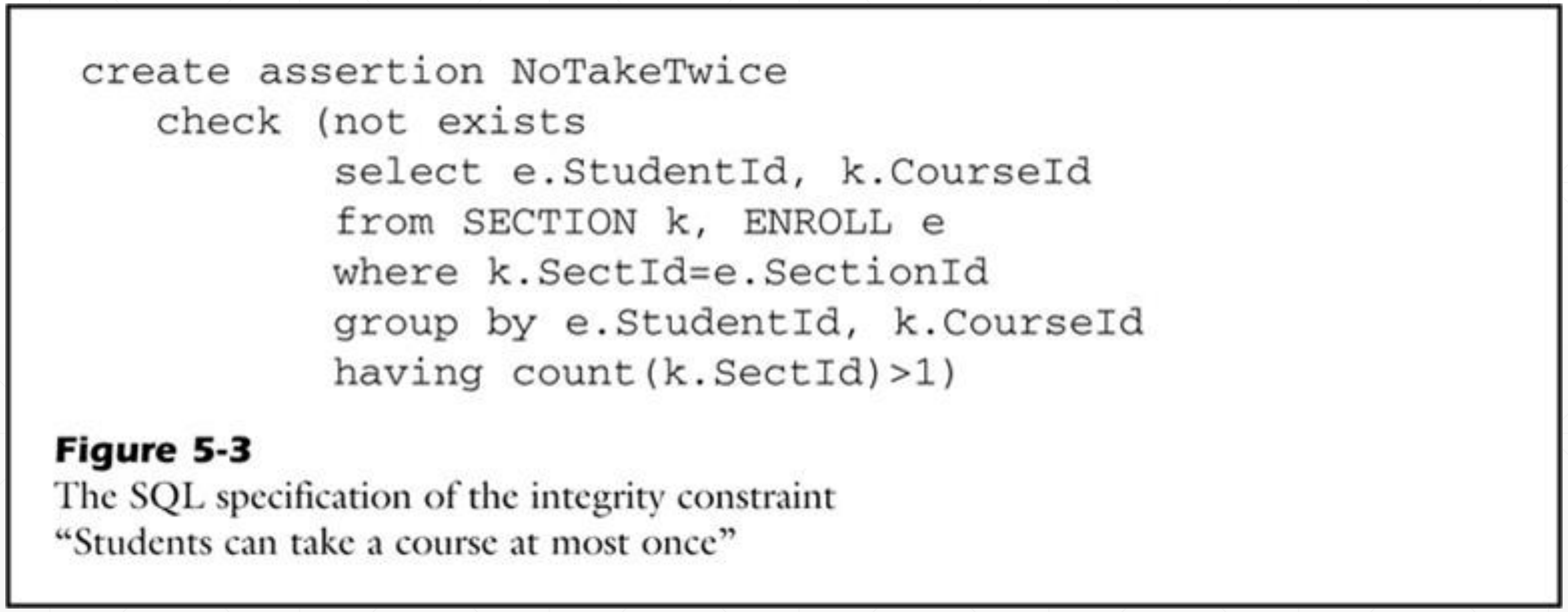


```
create assertion ValidGradYear
   check (not exists
          select s.*
          from STUDENT s
          where s.GradYear < 1863)
```

Figure 5-2
The SQL specification of the integrity constraint
“A student’s graduation year must be at least 1863”

Assertions (Cont.)

- The previous two assertions applied to a single table.
- Figure 5-3 gives an example of an assertion that applies to multiple tables.



```
create assertion NoTakeTwice
   check (not exists
          select e.StudentId, k.CourseId
          from SECTION k, ENROLL e
          where k.SectId=e.SectionId
          group by e.StudentId, k.CourseId
          having count(k.SectId)>1)
```

Figure 5-3
The SQL specification of the integrity constraint
“Students can take a course at most once”

Assertions in Oracle!

- There is no CREATE ASSERTION in Oracle.
  - DBMS do not necessarily implement all features in the SQL standards.

Triggers

- A trigger specifies an action that the database system invokes automatically whenever a specified event occurs.
- A trigger has three essential parts:
  - an ***event***: the update statement that initiates the activity
  - a ***condition***: the predicate that determines whether the trigger will fire
  - an ***action***: what happens if the trigger does fire

Triggers (Cont.)

```
CREATE TRIGGER trigger_name
        {BEFORE | AFTER} {INSERT | DELETE | UPDATE} [OF column_name] ON tableName
        [REFERENCING <old or new values alias list>]
        [FOR EACH ROW]
        [WHEN condition]
        action
```

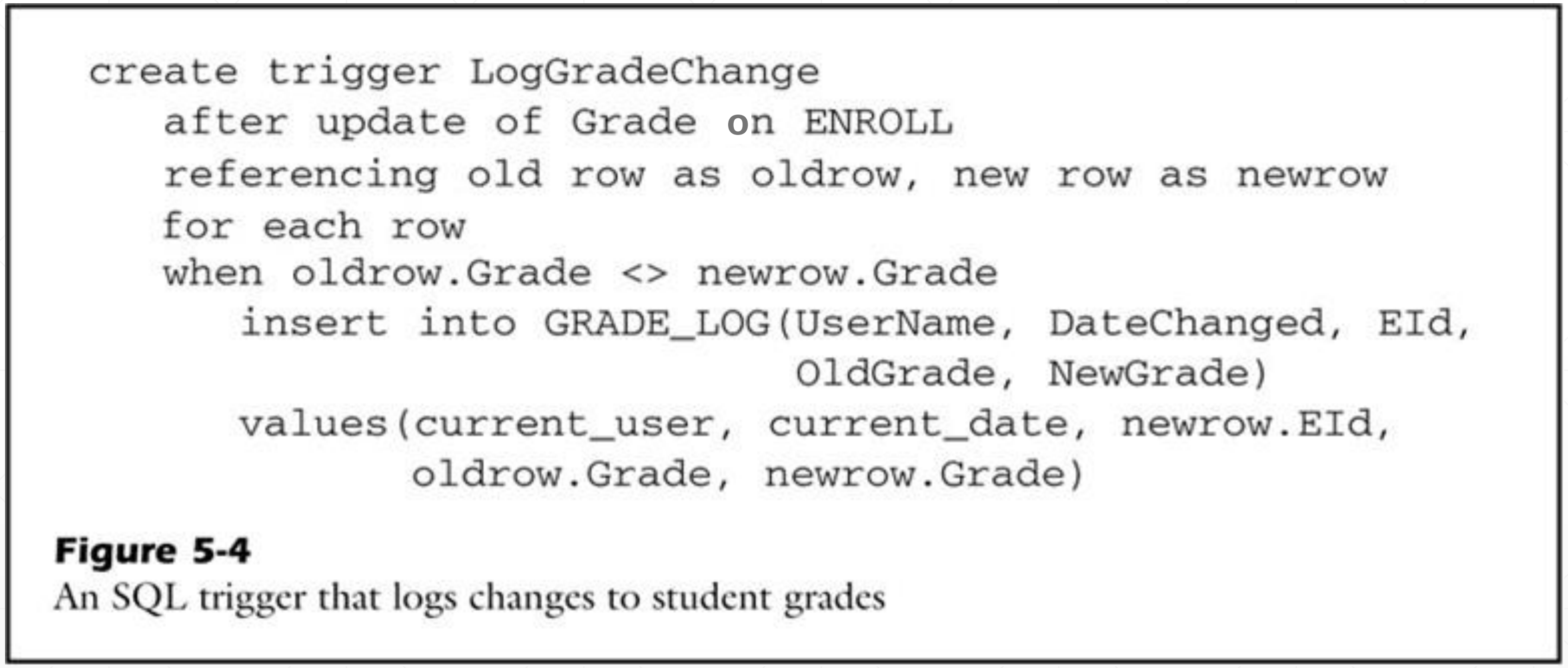
triggering_event

The action will apply to each record modified by the statement.

A trigger action can consist of several update statements; these statements would be bracketed by the keywords BEGIN and END.

- Implementations vary significantly.

Triggers (Cont.)



```
create trigger LogGradeChange
   after update of Grade on ENROLL
   referencing old row as oldrow, new row as newrow
   for each row
   when oldrow.Grade <> newrow.Grade
      insert into GRADE_LOG(UserName, DateChanged, EId,
                             OldGrade, NewGrade)
      values(current_user, current_date, newrow.EId,
             oldrow.Grade, newrow.Grade)
```

Figure 5-4
An SQL trigger that logs changes to student grades

Triggers (Cont.)

- Example: Consider a SECTION record that has exactly one ENROLL record connected to it. Write a trigger to delete that SECTION record when that ENROLL record is deleted.

```
create trigger DeleteEmptySection
      after delete on ENROLL
      referencing old row as oldrow
      for each row
            delete from SECTION
            where SectId = oldrow.SectionId
            and SectId not in (select SectionId from ENROLL)
```

Triggers in Oracle

triggering_event

```
CREATE [OR REPLACE] TRIGGER trigger_name
        {BEFORE | AFTER} {INSERT | DELETE | UPDATE} [OF column_name] ON
        tableName
        [REFERENCING old as oldrow  new as newrow]
        [FOR EACH ROW]
        [ENABLE / DISABLE]
        [WHEN condition]
        BEGIN
                executable statements
        END;
```

Triggers in Oracle

```
CREATE [OR REPLACE] TRIGGER trigger_name
        {BEFORE | AFTER} triggering_event ON tableName
        [REFERENCING old as oldrow  new as newrow]
        [FOR EACH ROW]
        [ENABLE / DISABLE]
        [WHEN condition]
        BEGIN
                executable statements
        END;
```

- New values of columns
  :new.column-name
- Existing values of columns
  :old.column-name

Triggers in Oracle (Cont.)

- A trigger can be enabled and disabled.
- While a trigger is enabled, the database automatically invokes it.
  - The trigger fires — whenever its triggering event occurs
- While a trigger is disabled, it does not fire.

Triggers in Oracle (Cont.)

```
CREATE OR REPLACE TRIGGER CommUpdate
        BEFORE UPDATE OF sal ON emp
        FOR EACH ROW
        WHEN (new.sal >3000)
        BEGIN
                :new.comm := :old.sal / 20;
        END;
```

- :new for new version of the value
- :old and old version of the value

Triggers in Oracle (Cont.)

```
CREATE OR REPLACE TRIGGER DeleteDept
        BEFORE DELETE ON dept
        FOR EACH ROW
        BEGIN
                delete from emp
                where deptno = :old.deptno;
        END;
```

How Triggers and Constraints Differ

- Both triggers and constraints can constrain data input.
- Constraints are easier to write and less error-prone than triggers that enforce the same rules.
- Triggers can enforce some complex business rules that constraints cannot.
- Triggers can be used to implement certain integrity constraints that cannot be specified using the constraint mechanism of SQL.
- A constraint rejects any update that violates it.
- A trigger action can fix the bad update.

Authorization

- Constraints and triggers help keep the database from becoming corrupted, by catching well-intentioned but problematic updates.
- We now consider
  - how to keep malicious users from corrupting the database intentionally
  - how to keep overly curious users from looking at private data

Authorization: Privileges

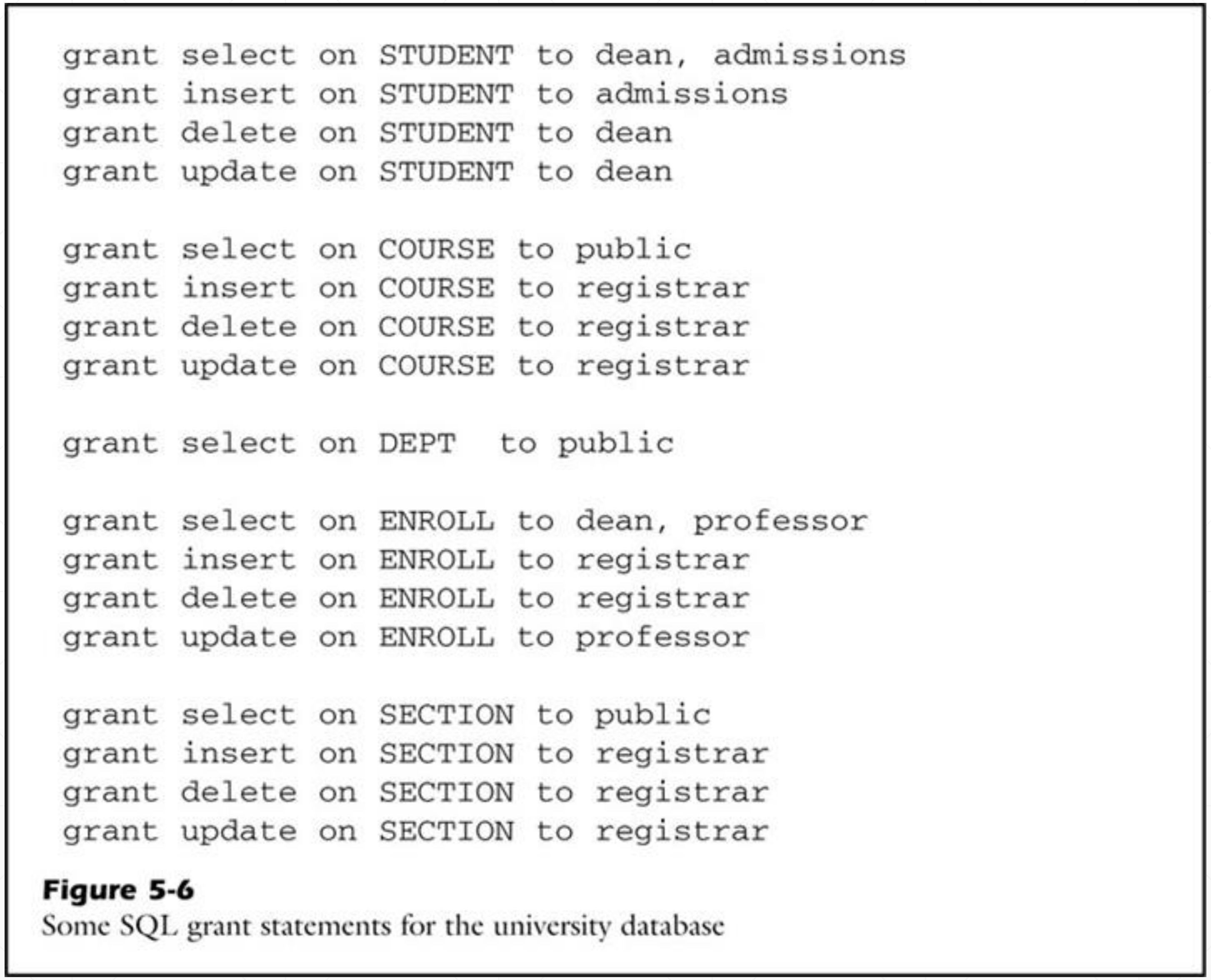
- In SQL, users can access the database only if they have been granted the appropriate privileges.
- The creator of a table grants privileges on it to other users.
- SQL has several kinds of privileges such as
  - select
  - insert
  - delete
  - update

Authorization: Privileges (Cont.)

- Example: grant insert on SECTION to damla
  - Assigns an insert privilege on the SECTION table to the user “damla”
- Example: grant select on COURSE to public
  - Assigns a select privilege on the COURSE table to all users

Authorization: Users and Roles

- If we assume that each professor logs into the database individually:
  - grant update on ENROLL to einstein, newton, brando,... (true) (feasible?)
- SQL avoids difficulty by distinguishing between users and roles.
  - A user is the person who is logged into the database.
  - A role is a category of users.
- A privilege can be granted to a user individually, or to a role.
  - In a large database environment (such as university), it is often used to create a role for each job function, and to grant privileges to roles.



```
grant select on STUDENT to dean, admissions
grant insert on STUDENT to admissions
grant delete on STUDENT to dean
grant update on STUDENT to dean

grant select on COURSE to public
grant insert on COURSE to registrar
grant delete on COURSE to registrar
grant update on COURSE to registrar

grant select on DEPT  to public

grant select on ENROLL to dean, professor
grant insert on ENROLL to registrar
grant delete on ENROLL to registrar
grant update on ENROLL to professor

grant select on SECTION to public
grant insert on SECTION to registrar
grant delete on SECTION to registrar
grant update on SECTION to registrar
```

Figure 5-6
Some SQL grant statements for the university database

Authorization: Users and Roles (Cont.)

- Roles in Figure 5-6: dean, admissions, registrar, professor
  - Each professor is assigned to the “professor” role, so on.
- A user having more than one job function will be assigned to multiple roles.
  - The privileges will be the union of the privileges of both roles.

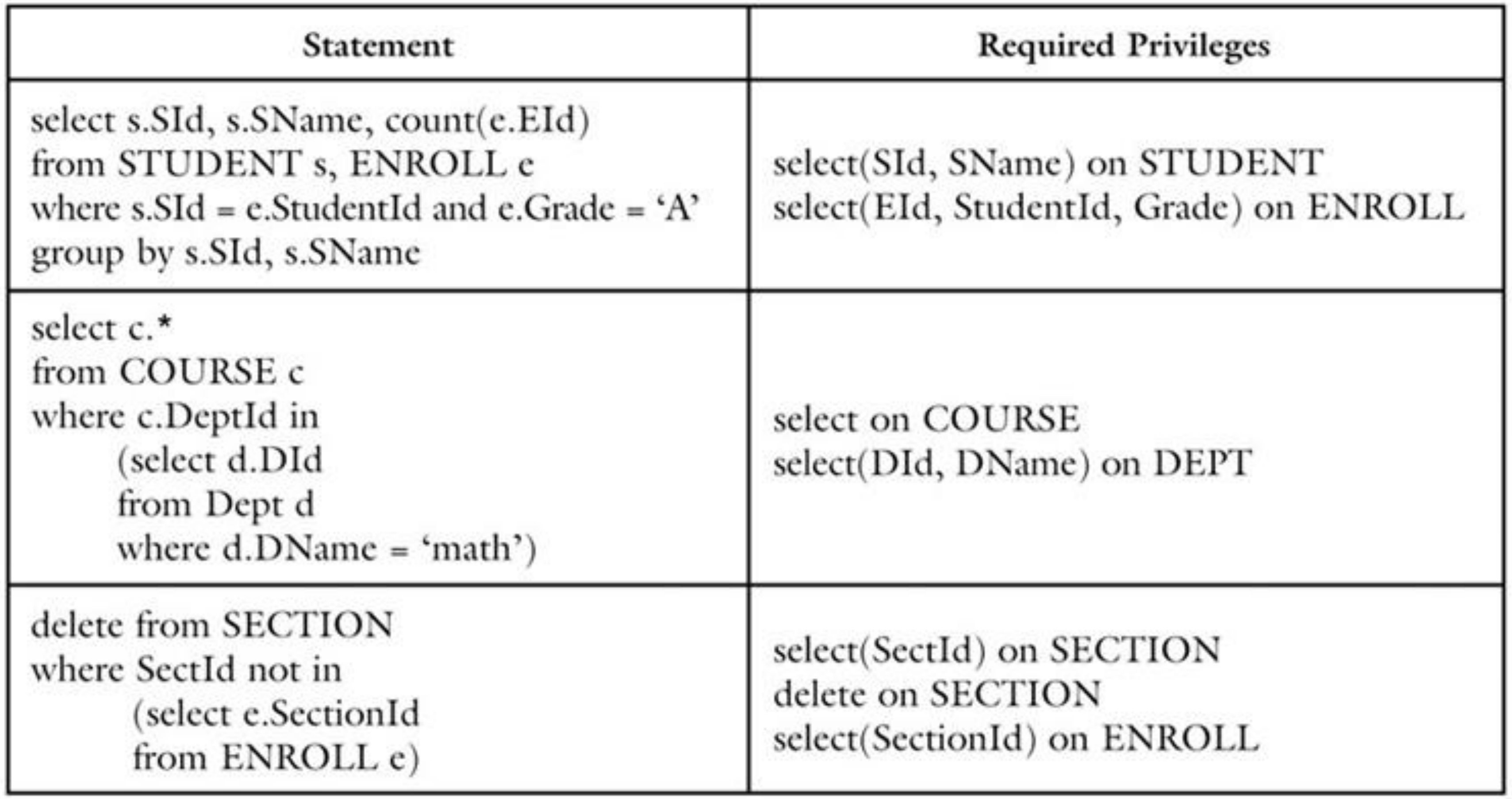
Authorization: Column Privileges

- SQL allows privileges to be granted on a specific columns of a table.
- The following statements keep the values of *StudentId* and *Grade* private, but make *EId* and *SectionId* public:
  - grant select (StudentId, Grade) on ENROLL to dean, professor
  - grant select (EId, SectionId) on ENROLL to public
- Column privileges can also be specified for update and insert operations.
  - grant update (Grade) on ENROLL to professor
  - grant insert (SName, MajorId) on STUDENT to admissions

Authorization: Statement Authorization

- A user is authorized to execute a statement if that user has all the privileges required by that statement.

Statement	Required Privileges
select	***select*** privilege on every mentioned field in the query
insert	***insert*** privilege for the fields to be inserted + a ***select*** privilege on every field in the ***where*** clause
update	***update*** privilege for the fields to be modified + a ***select*** privilege on every field in the ***where*** clause
delete	***delete*** privilege for the table + ***select*** privilege on every field in the ***where*** clause



Statement	Required Privileges
select s.SId, s.SName, count(e.EId) from STUDENT s, ENROLL e where s.SId = e.StudentId and e.Grade = 'A' group by s.SId, s.SName	select(SId, SName) on STUDENT select(EId, StudentId, Grade) on ENROLL

select c.*
from COURSE c
where c.DeptId in
    (select d.DId
    from Dept d
    where d.DName = 'math')
	select on COURSE
	select(DId, DName) on DEPT

delete from SECTION
where SectId not in
    (select e.SectionId
    from ENROLL e)
	select(SectId) on SECTION
	delete on SECTION
	select(SectionId) on ENROLL

Figure 5-7
Some SQL statements and their required privileges

Authorization: Privileges on Constraints

- grant references on STUDENT to dean
  - Dean’s office is allowed to create constraints that mention the STUDENT table

Authorization: Grant-Option Privileges

- The creator of the table
  - has privileges on the table
  - is able to grant these privileges to others
  - can grant a privilege to grant a privilege on the table
    - grant-option privilege

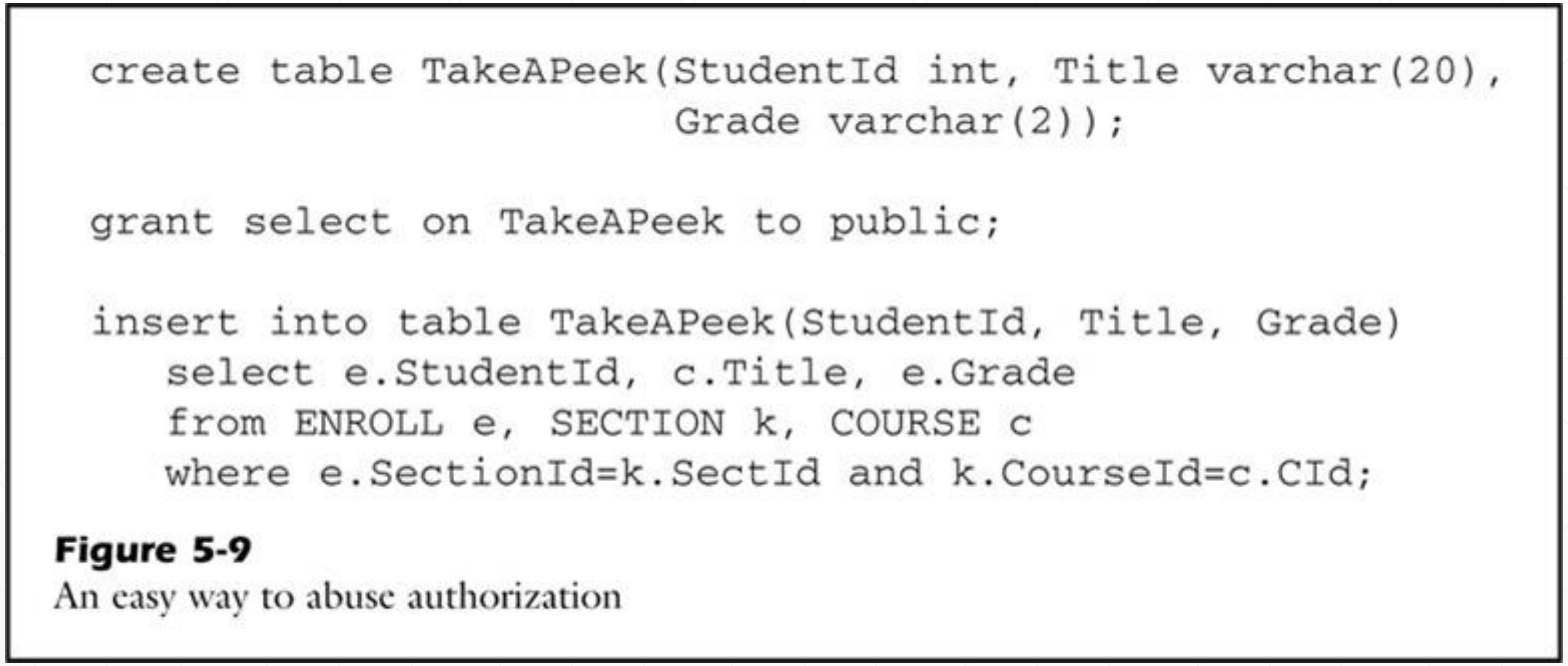
Authorization: Grant-Option Privileges (Cont.)

- Example: grant insert on STUDENT to admissions with grant option
  - the privilege to insert records into STUDENT
  - the privilege to grant that privilege to others
- The grant-option privilege is very powerful.

Mandatory Access Control

- grant select (StudentId, Grade) on ENROLL to dean, professor
- grant select (EId, SectionId) on ENROLL to public
- The professor can create a table and insert the confidential grade information into it.
- Since professor is the creator of the table, she/he can grant option privileges on it.
  - Can authorize anybody to look at it.
- The SQL authorization mechanism is called discretionary access control.
  - The creator of a table to authorize access to it, at his discretion.
- A database system that uses discretionary access control depends on its users to be trustworthy.

Mandatory Access Control (Cont.)



```
create table TakeAPeek(StudentId int, Title varchar(20),
                       Grade varchar(2));

grant select on TakeAPeek to public;

insert into table TakeAPeek(StudentId, Title, Grade)
   select e.StudentId, c.Title, e.Grade
   from ENROLL e, SECTION k, COURSE c
   where e.SectionId=k.SectId and k.CourseId=c.CId;
```

Figure 5-9
An easy way to abuse authorization

Mandatory Access Control (Cont.)

- Another option is to assign privileges to data, instead of tables.
- If a user inserts confidential data from the ENROLL table into another table, then data will remain confidential.
- This mechanism is called mandatory access control.
- Each record is assigned a classification level.
- Each user is assigned to a classification level.
- A user is authorized to see the record if the user’s classification level is at least as high as that of the record.

Mandatory Access Control (Cont.)

- The purpose of mandatory access control is to improve the security of a database system.
- However, commercial databases rarely use this mechanism.
  - Oracle Label Security, an add-on security option for the Oracle Enterprise Edition, enables you to customize your own label-based access control policies.
- It is suited primarily for high-security systems, such as in the military.

References

- Edward Sciore, Database Design and Implementation, Wiley, 2009.
  - Chapter 5
- https://docs.oracle.com/en/database/oracle/oracle-database/19/