Bancos de Dados

Conceitos Avançados

Prof. Filipe Mutz

Agenda

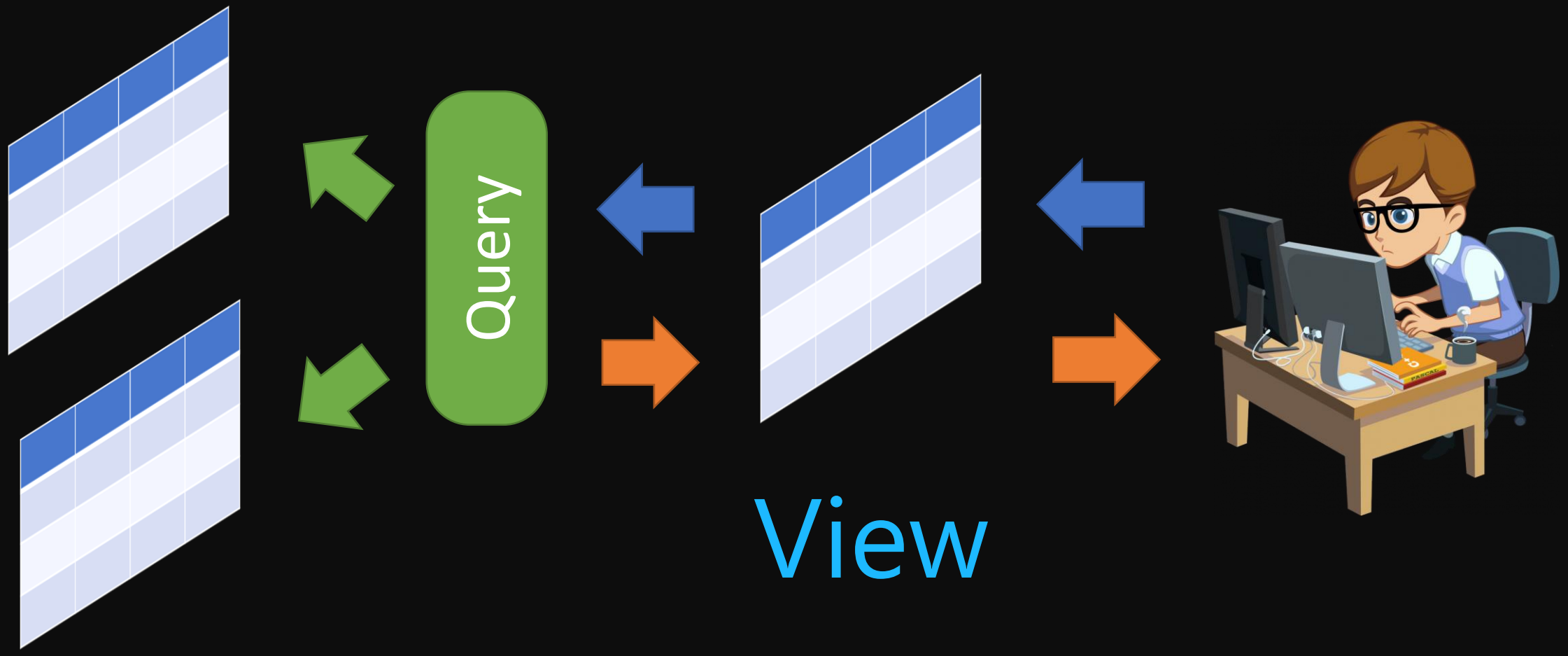
- Views
- Authorization
- Transactions
- Assertions
- Stored Procedures
- Triggers

Views

- Visões sobre partes do BD.
- Definidas por uma Query.
- Se Comportam como Tabelas.
- Podem ser usadas em Consultas.
- Em geral, **não** são armazenadas e são read-only.

Views: Motivação

- Esconder dados por Segurança ou Privacidade.
- Visualização simplificada para um grupo de usuários.
- Reduzir duplicação de partes de consultas frequentes.



View

Tabelas
"Verdadeiras"

Usuário		
E-mail	**Nome**	**CPF**
jose@gmail.com	José Silva	123123-123
luna@bol.com	Luna Vergara	456456-456

Empréstimo	
CPF	**codLivro**
123123-123	1
123123-123	2
456456-456	2

```
SELECT CPF, COUNT(codLivro) AS nEmprestimos FROM Usuario
LEFT OUTER JOIN Emprestimo ON Usuario.CPF = Emprestimo.CPF GROUP BY Usuario.CPF;
```

View ContadorEmprestimos	
CPF	**nEmprestimos**
123123-123	2
456456-456	1

Criação de Views

```
CREATE VIEW DEP_INFO(Dep_nome, Qtd_
            func, Total_sal)
AS SELECT   Dnome, COUNT (*), SUM
            (Salario)
    FROM    DEPARTAMENTO, FUNCIONARIO
    WHERE   Dnumero=Dnr
    GROUP BY Dnome;
```

Views: Implementação

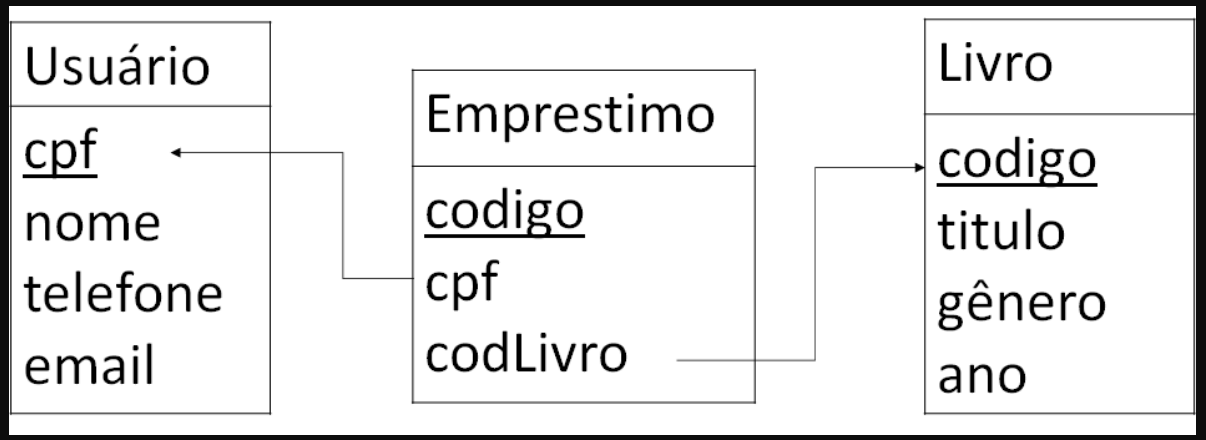
- Consulta realizada *on-demand*
  - Ineficiente se queries são complexas
- Materialização de Views
  - Usa espaço em disco
  - Requer mecanismos eficientes de lidar com atualizações nas tabelas de base.

Atualização de Views

In general, an SQL view is said to be **updatable** (that is, inserts, updates or deletes can be applied on the view) if the following conditions are all satisfied by the query defining the view:

- The **from** clause has only one database relation.
- The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
- Any attribute not listed in the **select** clause can be set to *null*; that is, it does not have a **not null** constraint and is not part of a primary key.
- The query does not have a **group by** or **having** clause.

```
CREATE VIEW  DEP_INFO(Dep_nome, Qtd_
             func, Total_sal)
AS SELECT    Dnome, COUNT (*), SUM
             (Salario)
    FROM     DEPARTAMENTO, FUNCIONARIO
    WHERE    Dnumero=Dnr
    GROUP BY Dnome;
```



Exercício: Crie uma *view* com títulos dos livros não devolvidos e o nome e telefone dos usuários que fizeram os empréstimos.

Autorização

- Criação de usuários
- Controle de Acesso
- Gerência de perfis
- Adição e remoção de privilégios (permissões)

Adição de Privilégios

grant <privilege list>
on <relation name or view name>
to <user/role list>;

grant select on *department* **to** *Amit, Satoshi;*

Amit e Satoshi podem realizar *selects* na tabela *department*

grant update (*budget*) **on** *department* **to** *Amit, Satoshi;*

Amit e Satoshi podem atualizar *budget* em *department*

Permissão para Inserir em Atributos

- Inserções devem especificar apenas esses atributos;
- O sistema fornece aos atributos restantes valores padrão (se um padrão for definido) ou *null*.

Remoção de Privilégios

revoke <privilege list>
on <relation name or view name>
from <user/role list>;

revoke select on *department* **from** *Amit, Satoshi;*
revoke update (budget) **on** *department* **from** *Amit, Satoshi;*

Papéis (Roles)

- Instrutores devem ter os mesmos privilégios.
- Para cada novo instrutor, permissões serão atribuídas individualmente.
- Uma abordagem melhor:
  - especificar os privilégios as autorizações que cada instrutor deve receber;
  - identificar quais usuários do banco de dados são instrutores.

```
create role instructor;
```

```
grant select on takes
to instructor;
```

```
create role dean;
grant dean to Amit;
grant instructor to dean;
grant dean to Satoshi;
```

Permissões de um papel atribuídas à outro papel.
O *dean* terá todos os privilégios do *instructor*.

Privilégios em *Views*

- Ex.: Um usuário deve poder ver os salários de um departamento, mas não de outros.
- Views podem ser usadas para limitar a visibilidade de dados. O usuário pode receber permissão de *SELECT* apenas na *view*.

```
create view geo_instructor as
    (select *
     from instructor
     where dept_name = 'Geology');
```

grant select on *geo_instructor* **to** *Marcus*;

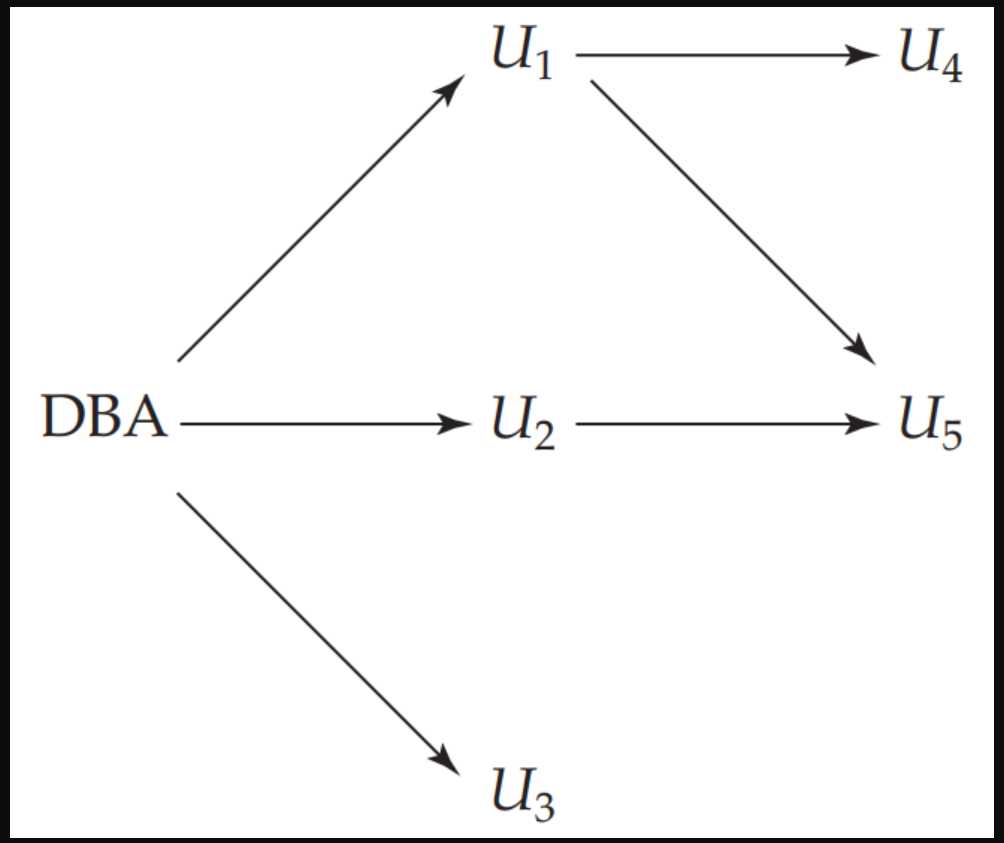
Transferência de Privilégios

grant select on *department* **to** Amit **with grant option;**

Amit pode realizar selects e dar esta permissão para outros usuários

Por padrão, isto não pode ser feito sem o “with grant option”, exceto para o usuário que criou o objeto (relação, view, role).

Grafo de Autorizações



- DBA autorizou U1, U2 e U3 com grant option.
- Em seguida, U1 autorizou U4 e U5, e U2 autorizou U5.

Revocação em Cascata

- DBA revoga a autorização de U1.
- U4 tem autorização do U1, então essa autorização também deve ser revogada.
- U5 recebeu autorização tanto do U1 quanto do U2. Como o DBA não revogou a autorização de U2, U5 retém os privilégios.
- Se U2 revogar a autorização de U5, ou o DBA revogar a autorização de U2, U5 perde os privilégios.

Revocação Restrita

revoke select on *department* **from** Amit, Satoshi **restrict;**

Nesse caso, o sistema retorna um erro se houver revogações em cascata e não executa a ação de revogação

Revocação de *Grant Option*

revoke grant option for select on *department* **from** Amit;

A instrução revoga apenas a opção de concessão de privilégios, e não o privilégio de realizar *SELECTS*

Exercício

- Remova todos os privilégios dos usuários Aurélio e Joubert.
- Crie o papel de Técnico.
- Dê todos os privilégios na tabela livro ao papel de Técnico.
- Dê privilégio de leitura na view criada no exercício anterior ao Aurélio com *grant option*.
- Atribua o papel de Tecnico a Joubert.

Transactions

- São operações com 1 ou mais passos que precisam ser executadas de forma atômica (indivisível).
  - Saque em uma conta e depósito em outra durante uma transferência.

Transactions

- Terminam de uma de duas formas:
- **COMMIT**: confirma a transação e faz com que as atualizações se tornem permanentes no banco de dados.
- **ROLLBACK**: faz com que a transação atual seja revertida e todas as atualizações realizadas são revertidas. O estado do banco de dados é restaurado ao que estava antes da execução da primeira instrução da transação.

Transactions

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO FUNCIONARIO (Pnome, Unome, Cpf, Dnr, Salario) VALUES ('Roberto', 'Silva', '99100432111', 2, 35.000);
EXEC SQL UPDATE FUNCIONARIO
    SET Salario = Salario * 1.1 WHERE Dnr = 2;
EXEC SQL COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ... ;
```

Transactions

- Geram problemas sofisticados devido à possibilidade de acessos concorrentes ao banco.
- Não entraremos em detalhes neste curso.

Assertions

- São CHECKS mais sofisticados e que podem envolver múltiplas tabelas.
- Permitem garantir a consistência entre dados.
  - Gasto com salários de um depto é igual à soma de salários dos funcionários.
  - Quantidade em estoque é igual à quantidade comprada menos a quantidade vendida.

Assertions

```
create assertion credits_earned_constraint check
(not exists (select ID
             from student
             where tot_cred <> (select sum(credits)
             from takes natural join course
             where student.ID= takes.ID
                   and grade is not null and grade<> 'F' )
```

Para cada tupla na relação aluno, o valor do atributo tot_cred deve ser igual à **soma dos créditos** das disciplinas que o aluno **concluiu com sucesso**.

Assertions

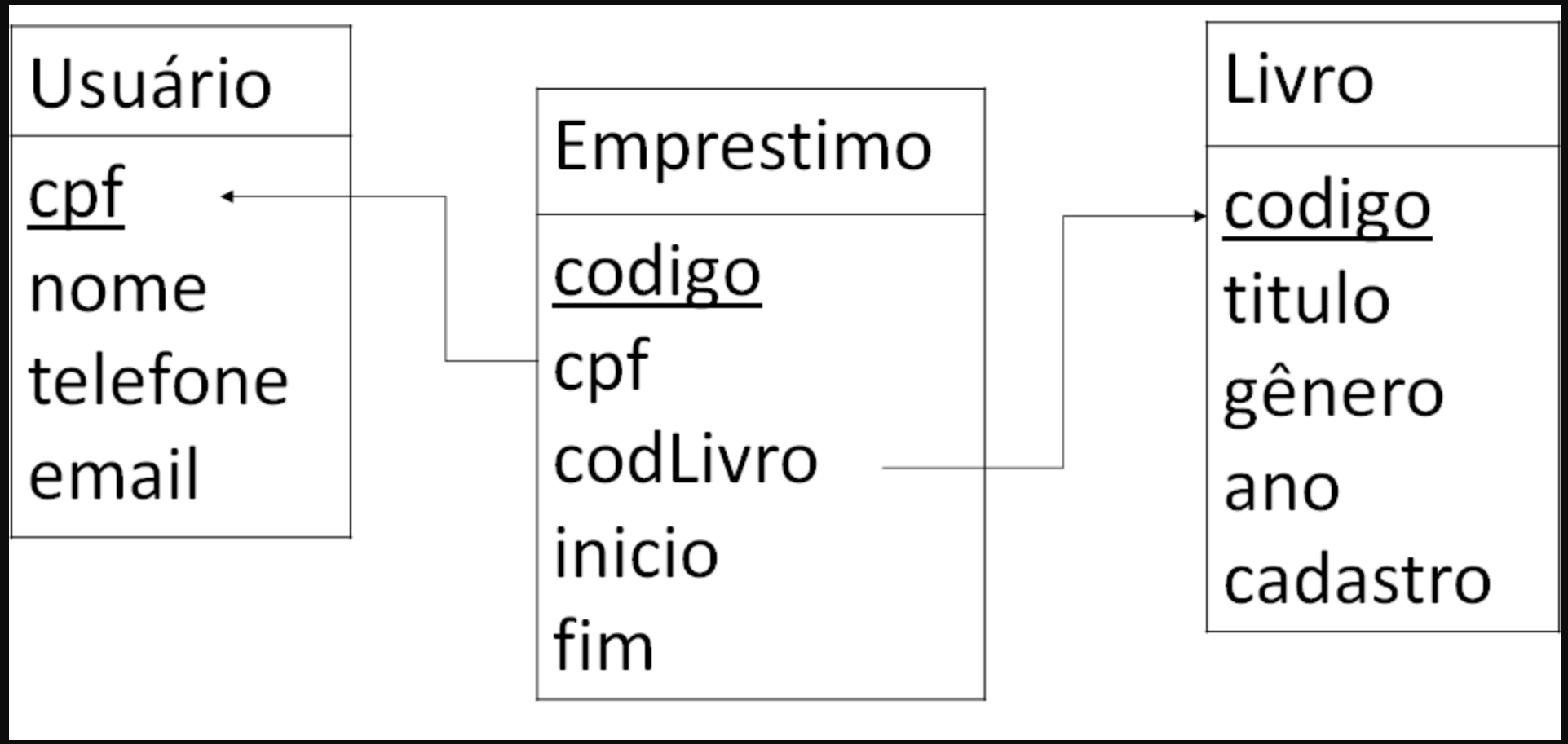
Processo de construção:

1. Faça um SELECT que retorne as tuplas que quebram a regra.
2. Use o comando NOT EXISTS para verificar que o retorno do SELECT é vazio (isto é, ninguém quebra a regra).

Assertions

- São avaliadas a cada alteração no BD.
- Se forem complexas e em grande quantidade, podem impactar negativamente o tempo de resposta do BD .
- **Não** são suportados por vários SGBDs.

Exercício: Crie uma assertion para verificar que a data de empréstimo e devolução dos livros é anterior à data de cadastro do livro no sistema.



Exercícios

[assuma o BD do exercício da semana passada inspirado no Google Academico]

- Crie um mecanismo para permitir que usuários da classe pesquisador vejam apenas artigos publicados no periódico Nature depois de 2015.
- Crie um mecanismo para permitir que usuários da classe pesquisador vejam o número de citações de autores da mesma instituição.
- Crie uma asserção para verificar que todos os artigos citados por um trabalho são anteriores ao trabalho.

Stored Procedures

- SGBDs podem armazenar **funções** (quando um valor é retornado) ou **procedimentos** (quando um valor não é retornado) similares àquelas de linguagens de programação.
- Com frequência, SGBDs comerciais permitem que procedimentos armazenados e funções sejam escritos em uma linguagem de programação de uso geral (C/C++, Java, etc.).

Quando são Úteis

- Se um programa de banco de dados é necessário por várias aplicações. Isso reduz a duplicação e melhora a modularidade.
- A execução de um programa no servidor pode reduzir a transferência de dados e o custo de comunicação entre o cliente e o servidor.
- Podem melhorar o poder de modelagem fornecido pelas visões ao permitir que tipos mais complexos de dados derivados estejam disponíveis aos usuários.
- Podem ser usados para verificar restrições complexas que estão além do poder de especificação de asserções e triggers

```
create function dept_count(dept_name varchar(20))
        returns integer
        begin
        declare d_count integer;
                select count(*) into d_count
                from instructor
                where instructor.dept_name= dept_name
        return d_count;
        end
```

```
select dept_name, budget
from instructor
where dept_count(dept_name) > 12;
```

Table Functions

```
create function instructors_of (dept_name varchar(20))
    returns table (
        ID varchar (5),
        name varchar (20),
        dept_name varchar (20),
        salary numeric (8,2))
return table
    (select ID, name, dept_name, salary
    from instructor
    where instructor.dept_name = instructor_of.dept_name);
```

```
select * from table(instructor of('Finance'));
```

```
create procedure dept_count_proc(in dept_name varchar(20),
                                 out d_count integer)
    begin
        select count(*) into d_count
        from instructor
        where instructor.dept_name= dept_count_proc.dept_name
    end
```

```
declare d_count integer;
call dept_count_proc('Physics', d_count);
```

Procedimentos podem ser invocados a partir de outro procedimento SQL ou de um SQL embutido usando a instrução *call*.

```
if boolean expression
        then statement or compound statement
elseif boolean expression
        then statement or compound statement
else statement or compound statement
end if
```

A parte do padrão SQL que lida com essas construções é chamada de **Persistent Storage Module (PSM)**. Variáveis são declaradas usando **declare** e atribuições são feitas usando **set**. Uma instrução composta tem a forma begin ... end e pode conter várias instruções SQL.

```
declare n integer default 0;
for r as
            select budget from department
            where dept_name = 'Music'
do
      set n = n− r.budget
end for
```

```
while boolean expression do
        sequence of statements;
end while

repeat
        sequence of statements;
until boolean expression
end repeat
```

1

```
-- Registers a student after ensuring classroom capacity is not exceeded
-- Returns 0 on success, and -1 if capacity is exceeded.
create function registerStudent(
                    in s_id varchar(5),
                    in s_courseid varchar (8),
                    in s_secid varchar (8),
                    in s_semester varchar (6),
                    in s_year numeric (4,0),
                    out errorMsg varchar(100)
returns integer
begin
 (...)

end;
```

2

```
declare currEnrol int;
select count(*) into currEnrol
        from takes
        where course_id = s_courseid and sec_id = s_secid
                and semester = s_semester and year = s_year;
declare limit int;
select capacity into limit
        from classroom natural join section
        where course_id = s_courseid and sec_id = s_secid
                and semester = s_semester and year = s_year;
if (currEnrol < limit)
        begin
                insert into takes values
                        (s_id, s_courseid, s_secid, s_semester, s_year, null);
                return(0);
        end
-- Otherwise, section capacity limit already reached
set errorMsg = 'Enrollment limit reached for course ' || s_courseid
        || ' section ' || s_secid;
return(-1);
```

Triggers

- Instrução executada automaticamente sempre que uma dada ação é feita no BD.
- Requer especificar:
  - **Quando** executar: Evento + Condição
  - **O que** fazer: Ações
- Uma vez criado, sempre que o evento acontecer e a condição for satisfeita, o SGBD realiza as ações.

Exemplos

- Sempre que uma tupla for inserida na relação Matrícula, incrementa o número de créditos cursados na tabela Aluno.
- Criar pedidos automaticamente para garantir uma quantidade mínima de itens em estoque.
- Ao finalizar uma venda, reduzir a quantidade de itens em estoque.

```
CREATE TRIGGER log_trigger
AFTER INSERT ON funcionarios
FOR EACH ROW
EXECUTE PROCEDURE funcionario_log_func();
```

A cada INSERT na tabela *funcionarios*, é executado o procedimento *funcionario_log_func()*. Ele poderia ser usado para salvar os dados do funcionário em uma tabela de log.

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> 'F' and nrow.grade is not null
    and (orow.grade = 'F' or orow.grade is null)
begin atomic
    update student
    set tot_cred= tot_cred+
            (select credits
             from course
             where course.course_id= nrow.course_id)
    where student.id = nrow.id;
end;
```

Sempre que a nota for atualiza de “F” para uma nota maior, ou de *null* para uma nota diferente de “F”, o total de créditos do aluno é atualizado.

```
create trigger reorder 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 atomic
    insert into orders
        (select item, amount
         from reorder
         where reorder.item = orow.item);
end;
```

Adiciona uma ordem de compra quando a quantidade em estoque de um produto se tornar menor que um limiar.