Capítulo 6: Desenho de Bases de Dados

- Objectivos com o Desenho de Bases de Dados
- Dependências funcionais
- 1ª Forma Normal
- Decomposição
- Forma Normal de Boyce-Codd
- 3ª Forma Normal
- Dependências multivalor
- 4ª Forma Normal
- Visão geral sobre o processo de design

Dependências Multivalor

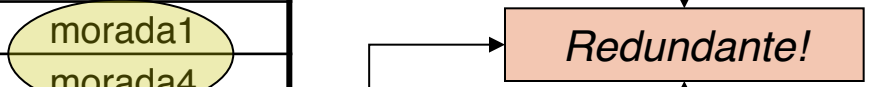
- Há bases de dados na BCNF que preservam as dependências, mas que não parecem estar suficientemente normalizadas
- Considere o seguinte esquema para o exemplo do banco:

  depositante(conta, nome, morada)
- Se tivermos *nome* → *morada* *(cada cliente tem apenas uma morada)* então este esquema não está na BCNF
- Mas o banco quer deixar que um cliente possa ter mais do que uma morada, i.e. não quer impor esta dependência funcional
- Nesse caso, *depositante* já está na BCNF.

Dependências Multivalor

depositante

conta	nome	morada
1	Carlos	morada1
1	Carlos	morada2
1	José	morada3
2	Carlos	morada1
2	Carlos	morada2
2	Maria	morada1
2	Maria	morada4
3	José	morada3
3	Maria	morada1
3	Maria	morada4



- Como não há dependências não triviais, (*conta, nome, morada*) é a única chave, e logo está na BCNF
- Mas:
  - Ainda existe redundância.
  - Problemas na inserção – Se quisermos adicionar uma nova morada (morada5) para o José, é necessário introduzir 2 tuplos:

    (1, José, morada5)

    (3, José, morada5)

Dependências Multivalor

- Parece melhor decompor em:

titulares

conta	nome
1	Carlos
1	José
2	Carlos
2	Maria
3	José
3	Maria

moradas

nome	morada
Carlos	morada1
Carlos	morada2
José	morada3
Maria	morada1
Maria	morada4

- Mas porquê? Que propriedades têm estes dados que permitem dizer que esta decomposição é apropriada? Como as exprimir?
- É certo que um dado cliente não têm sempre a mesma morada (independentemente da conta).
- Mas tem sempre o mesmo conjunto de moradas, independentemente da conta!

Dependências Multivalor

- Seja R um esquema, $\alpha \subseteq R$ e $\beta \subseteq R$. A *dependência multivalor*

$$\alpha \twoheadrightarrow \beta$$

é verdadeira em R se em toda a relação possível $r(R)$, para todo o par de tuplos t_1, t_2 em r, se $t_1[\alpha] = t_2[\alpha]$, então existem necessariamente tuplos t_3 e t_4, em r, tal que:

- $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
- $t_3[\beta] = t_1[\beta]$
- $t_3[R - \beta] = t_2[R - \beta]$
- $t_4[\beta] = t_2[\beta]$
- $t_4[R - \beta] = t_1[R - \beta]$

	α	β	R - α - β
t_1	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c_1 \ldots c_n$
t_2	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c'_1 \ldots c'_n$
t_3	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c'_1 \ldots c'_n$
t_4	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c_1 \ldots c_n$

Dependências Multivalor (Cont.)

$\alpha \twoheadrightarrow \beta$

	α	β	R - α - β
t_1	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c_1 \ldots c_n$
t_2	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c'_1 \ldots c'_n$
t_3	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c'_1 \ldots c'_n$
t_4	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c_1 \ldots c_n$

nome $\twoheadrightarrow$ morada

	nome	morada	conta
t_1	Ana	Lisboa	1
t_2	Ana	Coimbra	2
t_3	Ana	Lisboa	2
t_4	Ana	Coimbra	1

- Se dois tuplos (t1 e t2) têm o mesmo **nome**, tendo:
  - t1: **morada**=Lisboa e **conta**=1 e
  - t2: **morada**=Coimbra e **conta**=2
- Então têm que existir mais dois tuplos (t3 e t4) com o mesmo **nome**, tendo:
  - t3: **morada**=Lisboa e **conta**=2
  - t4: **morada**=Coimbra e **conta**=1

Dependências Multivalor

- Seja R um esquema, $\alpha \subseteq R$ e $\beta \subseteq R$. A *dependência multivalor*

$$\alpha \twoheadrightarrow \beta$$

é verdadeira em R se em toda a relação possível $r(R)$, para todo o par de tuplos t_1, t_2 em r, se $t_1[\alpha] = t_2[\alpha]$, então existem necessariamente tuplos t_3 e t_4, em r, tal que:

- $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
- $t_3[\beta] = t_1[\beta]$
- $t_3[R - \beta] = t_2[R - \beta]$
- $t_4[\beta] = t_2[\beta]$
- $t_4[R - \beta] = t_1[R - \beta]$

	α	β	R - α - β
t_1	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c_1 \ldots c_n$
t_2	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c'_1 \ldots c'_n$
t_3	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c'_1 \ldots c'_n$
t_4	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c_1 \ldots c_n$

Dependências Multivalor (Cont.)

$\alpha \twoheadrightarrow \beta$

	α	β	R - α - β
t_1	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c_1 \ldots c_n$
t_2	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c_1 \ldots c_n$
t_3	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c'_1 \ldots c'_n$
t_4	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c'_1 \ldots c'_n$

nome $\twoheadrightarrow$ morada

	nome	morada	conta
t_1	Ana	Lisboa	1
t_2	Ana	Coimbra	1
t_3	Ana	Lisboa	2
t_4	Ana	Coimbra	2

- Se existem dois tuplos (t1 e t2) com o mesmo **nome** e **conta**, tendo:
  - t1: **morada**=Lisboa e
  - t2: **morada**=Coimbra
- Se o cliente abrir uma nova conta, temos que adicionar dois tuplos com o mesmo nome e a nova conta, tendo:
  - t3: **morada**=Lisboa
  - t4: **morada**=Coimbra

Dependências Multivalor (Cont.)

$\alpha \twoheadrightarrow \beta$

	α	β	R - α - β
t_1	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c_1 \ldots c_n$
t_2	$a_1 \ldots a_i$	$b_1 \ldots b_j$	$c'_1 \ldots c'_n$
t_3	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c_1 \ldots c_n$
t_4	$a_1 \ldots a_i$	$b'_1 \ldots b'_j$	$c'_1 \ldots c'_n$

nome $\twoheadrightarrow$ morada

	nome	morada	conta
t_1	Ana	Lisboa	1
t_2	Ana	Lisboa	2
t_3	Ana	Coimbra	1
t_4	Ana	Coimbra	2

- Se existem dois (t1 e t2) com o mesmo **nome** e **morada**, tendo:
  - t1: **conta**=1 e
  - t2: **conta**=2
- Se o cliente passar a ter uma nova morada, temos que adicionar dois tuplos com o mesmo nome e a nova morada, tendo:
  - t3: **conta**=1 e
  - t4: **conta**=2

Propriedades

- Seja *R* um esquema com um conjunto de atributos particionados em 3 subconjuntos não vazios *Y, Z e W.*
- Diz-se que $Y \twoheadrightarrow Z$ *(Y* multidetermina *Z) s*se para todas as possíveis *r(R)*

	$(y_1, z_1, w_1) \in r$		$(y_1, z_1, w_2) \in r$
Se	e	Então	e
	$(y_1, z_2, w_2) \in r$		$(y_1, z_2, w_1) \in r$

- Note que, como esta definição é simétrica em Z e W, segue que

  $Y \twoheadrightarrow Z$ sse $Y \twoheadrightarrow W$ (i.e. $Y \twoheadrightarrow R\text{-}Y\text{-}Z$)

- Note ainda que:
  - Se $Y \rightarrow Z$ então $Y \twoheadrightarrow Z$
  - De facto, se $Y \rightarrow Z$ então $z_1 = z_2$, e logo $Y \twoheadrightarrow Z$.

Exemplo

- No nosso exemplo:

  nome $\twoheadrightarrow$ **morada**
  nome $\twoheadrightarrow$ **conta**

- Esta definição formaliza a ideia de que cada valor particular de *Y* (**nome**) tem associado um conjunto de valores *Z* (**morada**) e um conjunto de valores de *W* (**conta**), e que estes dois conjuntos são independentes.
- Se são independentes, porque não colocá-los em relações separadas?

Uso de Dependências Multivalor

- Usam-se dependências multivalor para:
  1. Testar relações, para verificar se são ou não relações válidas, dado um conjunto de dependências multivalor
  2. Especificar restrições no conjunto de (instâncias) de relações válidas. Assim, só devemos ter relações que satisfaçam o conjunto (pré-definido) de dependências funcionais e multivalor.
- Se uma relação r não satisfizer uma dada dependência multivalor, então é sempre possível construir uma relação r', por adição de tuplos em r, que satisfaça a dependência.

Teoria de Dependências Multivalor

- Da definição de dependência multivalor, pode-se demonstrar:
  - Se $\alpha \rightarrow \beta$, então $\alpha \twoheadrightarrow \beta$
    - I.e. toda a dependência funcional é também dependência multivalor.
  - $\alpha \twoheadrightarrow \beta$ é trivial sse $\beta \subseteq \alpha$ ou $\alpha \cup \beta = R$
- Em geral temos um conjunto *D* de dependências funcionais e dependências multivalor
- O **fecho** D^+ de *D* é o conjunto de todas as dependências funcionais e multivalor que são implicadas por *D*.
  - Pode calcular-se D^+ a partir de *D*, usando as definições de dependência funcional e multivalor.
  - Tal como para dependências funcionais, há sistemas de inferência par calcular este fecho.

Inferência com Dependências Multivalor

- Podem encontrar-se todas as dependências em D^+ por aplicação dos seguintes Axiomas (onde os primeiros 3 são os Axiomas de Armstrong) :
  - Se $\beta \subseteq \alpha$, então $\alpha \rightarrow \beta$ **(reflexividade)**
  - Se $\alpha \rightarrow \beta$, então $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(aumento)**
  - Se $\alpha \rightarrow \beta$, e $\beta \rightarrow \gamma$, então $\alpha \rightarrow \gamma$ **(transitividade)**
  - Se $\alpha \twoheadrightarrow \beta$ então $\alpha \twoheadrightarrow R - \beta - \alpha$ **(complemento)**
  - Se $\alpha \twoheadrightarrow \beta$, $\gamma \subseteq R$ e $\delta \subseteq \gamma$ então $\gamma\ \alpha \twoheadrightarrow \delta\beta$ **(multi-aumento)**
  - Se $\alpha \twoheadrightarrow \beta$, e $\beta \twoheadrightarrow \gamma$, então $\alpha \twoheadrightarrow \gamma - \beta$ **(multi-transitividade)**
  - Se $\alpha \rightarrow \beta$ então $\alpha \twoheadrightarrow \beta$ **(replicação)**
  - Se $\alpha \twoheadrightarrow \beta$, $\gamma \subseteq \beta$ e existe $\delta \subseteq R$ tal que $\delta \cap \beta = \{\}$ e $\delta \rightarrow \gamma$ então $\alpha \rightarrow \gamma$ **(coalescência)**
- Este conjunto de axiomas é coerente e completo.

4ª Forma Normal

- Um esquema R, com conjunto de dependência funcionais e multivalor D, está na 4FN se para toda a dependência multivalor $\alpha \twoheadrightarrow \beta \in D^+$, onde $\alpha \subseteq R$ e $\beta \subseteq R$, pelo menos uma das seguintes condições é verdadeira:
  - $\alpha \twoheadrightarrow \beta$ é trivial (i.e., $\beta \subseteq \alpha$ ou $\alpha \cup \beta = R$)
  - α é super chave de R (i.e., $\alpha \rightarrow R \in D^+$)
- Se um esquema está na 4FN também está na BCNF

Restrição de Dependências Multivalor

- A restrição do conjunto D a R_i é o conjunto de D_i com
  - Todas as dependências em D^+ que só contêm atributos de R_i
  - Todas as dependências multivalor:

    $\alpha \twoheadrightarrow (\beta \cap R_i)$

    onde $\alpha \subseteq R_i$ e $\alpha \twoheadrightarrow \beta \in D^+$
- Com dependências multivalor, a decomposição de R em R_1 e R_2 é sem perdas sse pelo menos uma das dependências abaixo pertence a D^+:
  - $R_1 \cap R_2 \twoheadrightarrow R_1 - (R_1 \cap R_2)$
  - $R_1 \cap R_2 \twoheadrightarrow R_2 - (R_1 \cap R_2)$

Algoritmo de Decomposição para 4NF

result: = {*R*};
done := false;
calcular D^+;
Seja D_i a restrição de D^+ a R_i

while (**not** *done*)
  if (existe esquema $\mathbf{R}_i \in$ *result* que não está na 4NF) **then**
    begin

      Seja $\alpha \twoheadrightarrow \beta$ não trivial e verdadeira em R_i tal que
        $\alpha \rightarrow R_i \notin D_i$, e $\alpha \cap \beta = \{\}$;
      result := (*result* - R_i) $\cup$ (R_i - β) $\cup$ (α, β);
    end
  else *done*:= true;

Nota: A decomposição é sem perdas

Exemplo

- $R = (A, B, C, G, H, I)$

  $D = \{ A \twoheadrightarrow B$

  $B \twoheadrightarrow HI$

  $CG \rightarrow H \}$

- R não está na 4FN pois $A \twoheadrightarrow B \in F$ e A não é superchave de R
- Decomposição

  a) $R_1 = (A, B)$ (R_1 está na 4NF. A única dep. em R_1 é trivial: $A \cup B$)

  b) $R_2 = (A, C, G, H, I)$ (R_2 não está na 4NF – na 3ª dep. CG não é chave)

  c) $R_3 = (C, G, H)$ (R_3 está na 4FN)

  d) $R_4 = (A, C, G, I)$ (R_4 não está na 4FN – Como $A \twoheadrightarrow B$ e $B \twoheadrightarrow HI$ logo $A \twoheadrightarrow HI \in D^+$, e $A \twoheadrightarrow I$ está na restrição a R_4)

  e) $R_5 = (A, I)$ (R_5 está na 4FN)

  f) $R_6 = (A, C, G)$ (R_6 está na 4FN)

- Resultado da decomposição: $\{R_1, R_3, R_5, R_6\}$

4FN e Preservação de Dependências

- Tal como a BCNF, a 4FN pode não preservar as dependências:
  - $R = (A, B, C, G, H, I)$ com $D = \{ A \twoheadrightarrow B,\ B \twoheadrightarrow HI,\ CG \rightarrow H \}$ foi decomposto em $\{(A, B), (C, G, H), (A, I), (A, C, G)\}$
  - A dependência $B \twoheadrightarrow HI$ não pode ser testada apenas numa destas relações.
- Aplicam-se aqui as mesmas soluções de compromisso que entre a BCNF e a 3FN:
  - Objectivos numa primeira fase:
    - 4FN.
    - Decomposição sem perdas.
    - Preservação de dependências.
  - Se tal não for possível, então há que optar por uma de
    - Não preservação de dependências
    - Alguma redundância
      - Tentar BCNF.
      - Se tal ainda não preserva dependências, normalizar para a 3FN

Mais Formas Normais

- As **dependências de junção** generalizam as multivalor
  - Dão origem à **forma normal projecção-junção (**PJNF) (também chamada de **5ª forma normal**)
- Uma classe ainda mais geral de restrições leva à **forma normal de domínio-chave**.
- Problemas com estas restrições muito gerais:
  - é dificil raciocínar sobre elas
  - não têm conjuntos coerentes e completos de regras de inferência.
- Logo, raramente são usadas

Visão global sobre o design

- Temos assumido que o esquema *R* é dado
  - *R* pode ter sido obtido ao passar um diagrama E-R para tabelas.
  - *R* pode ser uma única relação contendo *todos* os atributos de interesse para os dados (**relação universal**).
    - A normalização irá decompor *R* em relações mais pequenas.
  - *R* pode ser o resultado de algum design ad hoc.

Modelo ER e Normalização

- Quando o diagrama E-R está ***mesmo*** bem feito, as tabelas geradas em princípio nem necessitam de mais normalização.
- No entanto, na prática há diagramas ER imperfeitos que levam a que dependências que queremos impor não tenham o lado esquerdo como chave.
- E.g. Entidade *empregado* com atributos *cod_departamento* e *morada_dep*, e a dependência *cod_departamento* → *morada_dep*
  - Num bom design *departamentos* seria um outro conjunto de entidades

Desnormalização para Performance

- Para melhorar a performance, podemos querer usar esquema não normalizados
- E.g. mostrar *customer-name* juntamente com *account-number* e *balance* exige junção de *account* com *depositor*
- Alternativa 1: Usar relação desnormalizada que contém atributos de *account* e de *depositor*
  - execução mais rápida de perguntas
  - gasta mais espaço, e mais tempo para actualizações
  - é mais passível de erros
- Alternativa 2 (melhor): usar uma **materialized view** definida como:
  account ⋈ depositor
  - As vantagens e desvantagens são como na outra alternativa, com excepção de que esta não aumenta a possibilidade de erros.

Outros aspectos do design

- Alguns aspectos do design de bases de dados não são captados pela normalização
- Exemplos de mau design, a evitar:

  Em vez de *lucros*(*companhia, ano, valor*), usar

  - *lucros-2000, lucros-2001, lucros-2002*, etc., todas com esquema (*companhia, valor*).
    - Todas estas relações estão na BCNF. Mas dificulta a execução de perguntas sobre vários anos, e exige nova tabela todos os anos
  - *companhia_ano*(*companhia, lucros-2000, lucros-2001, lucros-2002*)
    - Também está na BCNF, mas também dificulta perguntas sobre vários anos, e exige novo atributo todos os anos.
    - É um exemplo de **crosstab**, onde valores para um atributo se transformam em nomes de atributos
    - Usado em folhas de cálculo e em ferramentas de análise de dados

Capítulo 7: Segurança

- Segurança e Autorizações

Segurança

- **Segurança** – ao contrário das restrições de integridade, que pretendiam proteger a base de dados contra estragos acidentais, a segurança preocupa-se com proteger a base de dados de estragos propositados.
  - A nível do sistema operativo
  - A nível da rede
  - A nível físico
  - A nível humano
  - A nível da base de dados
    - Mecanismos de autenticação e autorização para permitir acessos selectivos de (certos) utilizadores a (certas) partes dos dados

Autorizações

Diferentes formas de autorização em dados da bases de dados:

- **Autorização de leitura** – permite ler, mas não modificar dados.
- **Autorização de inserção** – permite inserir novos tuplos, mas não modificar tuplos existentes.
- **Autorização de modificação** – permite modificar tuplos, mas não apagá-los.
- **Autorização de remoção** – permite apagar tuplos

Autorizações (Cont.)

Diferentes formas de autorização, para alterar esquemas:

- **Autorização de index** – permite criar e apagar ficheiros de index.
- **Autorização de resources** – permite criar novas relações.
- **Autorização de alteração** – permite criar e apagar atributos duma relação.
- **Autorização de drop** – permite apagar relações.

Autorizações e Vistas

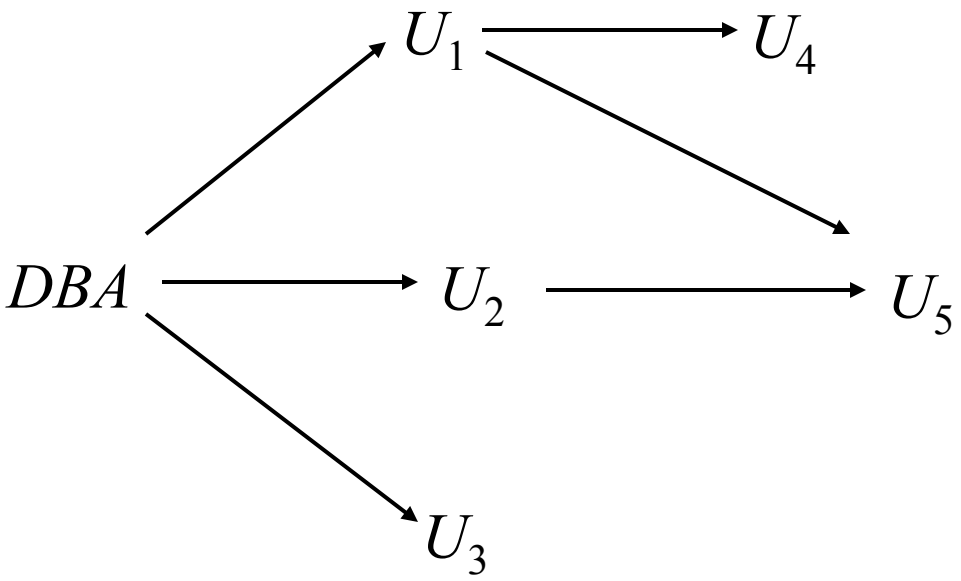
- Pode-se dar autorização a utilizadores sobre uma vista, sem se lhe dar autorização sobre as tabelas que a definem
- Isto permite não só melhorar a segurança dos dados, como também tornar mais simples o seu uso
- Uma combinação de segurança a nível de tabelas, com segurança a nível de vistas, pode ser usada para limitar o acesso de um utilizador apenas aos dados de que ele necessita.

Autorizações e Vistas

- A criação de uma vista não requer autorização **resources** pois, de facto, nenhuma nova tabela é criada
- Quem cria uma vista , fica exactamente com os mesmo privilégios sobre esta que tinha sobre as tabelas.
- E.g. o criador duma vista *cust_loan* sobre as tabelas *borrower* e *loan*, que só tenha autorização de leitura sobre estas tabelas, só fica com autorização de leitura sobre a vista que criou

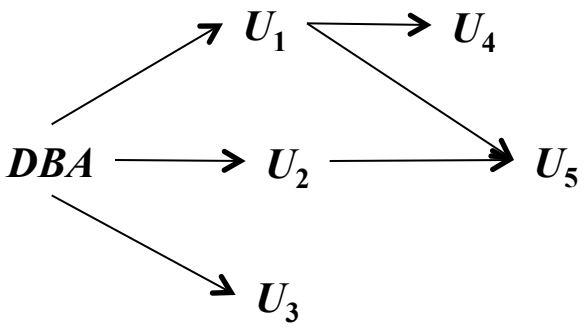
Atribuição de Privilégios

- A passagem de privilégios de um utilizador para outro pode ser representada por um grafo de autorizações.
- Os nós do grafo são utilizadores.
- A raiz é o administrador da base de dados.
- Considere o grafo abaixo, para e.g. escrita numa relação.
- Um arco $U_i \rightarrow U_j$ indica que o utilizador U_i atribuiu ao utilizador U_j privilégio de escrita sobre essa relação.

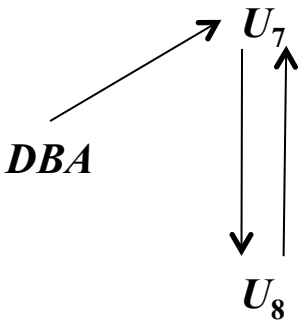


Grafo de atribuição de privilégios

- *Requisito*: Todos os arcos têm que fazer parte de algum caminho com origem no administrador.



- Se o administrados retira o privilégio a U_1:
  - Deve ser retirado privilégio a U_4 (pois U_1 já não tem autorização)
  - Não deve ser retirado a U_5 (pois U_5 tem autorização vinda de U_2)
- Devem ser prevenidos ciclos:
  - Administrador dá privilégios a U_7
  - U7 dá privilégios a U_8
  - U8 dá privilégios a U_7
  - DBA retira privilégios de U_7



- Deve retirar autorização de U_7 para U_8 e de U_8 para U_7 (pois já não há caminho do administrador nem para U_7 nem para U_8).

Especificações de Segurança em SQL

- O comando **grant** é usado para atribuir privilégios

  grant <lista de privilégios>

  on <nome de relação ou view> **to** <lista de utilizadores>

- <lista de utilizadores> é:
  - Um user-id
  - *public*, o que atribui o privilégios a todos os utilizadores
  - Um perfil (*role*) – veremos à frente
- A atribuição de privilégios sobre uma vista não se propaga às relações nela usadas.
- Quem atribui o privilégio tem que o ter (ou ser o administrador da base de dados).

Privilégios em SQL

- **select:** permite acesso de leitura sobre a relação ou vista
  - Exemplo: dar a U_1, U_2, e U_3 autorização de leitura **na relação** *branch*:

    grant select on *branch* **to** U_1, U_2, U_3
- **insert**: permite inserir tuplos
- **update**: permite usar o comando **update** do SQL
- **delete**: permite apagar tuplos.
- **references**: permite a declaração de chaves externas.
- **all privileges**: forma sumária de atribuir todos os privilégios.

Privilégio de atribuir privilégios

- **with grant option**: autoriza um utilizador a passar um privilégio a outros utilizadores.
  - Exemplo:

    grant select on *branch* **to** U_1 **with grant option**

    dá a U_1 o privilégio **select** sobre a relação *branch* e autoriza U_1 a passar esse privilégio a qualquer outro utilizador

Perfis

- Um perfil permite atribuir, de apenas uma vez, privilégios iguais para uma classe de utilizadores
- Podem ser atribuídos e retirados privilégios a perfis de utilizadores, da mesma forma que a utilizadores isolados.
- Podem-se associar perfis a utilizadores, ou mesmo a outros perfis
- Exemplo:

create role *caixa*
create role *gerente*

grant select on *branch* **to** *caixa*
grant update (*balance***) on** *account* **to** *caixa*
grant all privileges on *account* **to** *gerente*

grant *caixa* **to** *gerente*

grant *caixa* **to** *maria, scott*
grant *gerente* **to** *ana*

Retirar de privilégios em SQL

- O comando **revoke** serve para retirar privilégios.

  revoke <privilégios>

  on <relação ou view> **from** <utilizadores> [**restrict**|**cascade**]

- Exemplo:

  revoke select on *branch* **from** U_1, U_2, U_3 **cascade**

- Se se colocar **cascade**, retirar privilégios de um utilizador também os pode retirar a outros, conforme descrito pelo grafo.

- Se se usar **restrict** só é retirado privilégio a esse utilizador

  revoke select on *branch* **from** U_1 **restrict**

  Com **restrict**, o comando **revoke** falha (dá erro) se esse utilizador já passou o privilégio a outros.

Retirar de privilégios em SQL (Cont.)

- <privilégios> pode ser **all**. Nesse caso são retirados todos os privilégios *que foram atribuídos pelo utilizador* que deu o comando.
- Se <utilizadores> incluir **public** todos os utilizadores perdem esse privilégio, *a não ser que lhe tenha sido atribuído explicitamente*.
- Se o mesmo privilégio for atribuído duas vezes por utilizadores diferentes, então quem o tem pode ficar com ele mesmo depois dum **revoke** (cf. grafo).
- Todos os privilégios que dependem do privilégio retirado, são também retirados.

Limitação a autorizações em SQL

- SQL não permite autorizações a nível de tuplo
  - E.g. não se pode restringir de forma a que um aluno só possa ver as suas notas.
- Neste caso, a tarefa de autorização cai sobre as aplicações (o que é indesejável, mas o SQL aqui não ajuda).
- Ou então definir vistas e dar autorizações apenas a essas vistas