Bases de Dados

Parte VI
SQL em Ambiente de Programação

SQL em Ambiente de Programação

- O SQL pode ser usado de forma interactiva por utilização do interface normalmente disponibilizado pelo próprio SGBD (introduzir comando SQL/visualizar relação resultado). Este modo de operar pode ser conveniente na fase de criação da BD ou ocasionalmente para executar certas consultas.
- Na prática, a quase totalidade das interacções com uma BD é feita por aplicações ou interfaces externas desenhadas para esse efeito. O SQL não é usado directamente pelo utilizador, mas sim embebido na própria aplicação ou interface.
- O utilizador apenas interage com a aplicação ou interface externa. Estes por sua vez é que geram as expressões SQL que definem as acções pretendidas e tratam de as executar no SGBD.
- Para que tal seja possível, é necessário que a linguagem de programação na qual a aplicação/interface foi desenvolvida possua algum tipo de suporte para comandos SQL.

SQL em Ambiente de Programação

- A sequência de interacção básica com um SGBD é a seguinte:
  - Estabelecer a ligação com o SGBD (máquina, login, password, ...).
  - Enviar comandos SQL para manipular ou obter informação da BD.
  - Terminar a ligação com o SGBD.
- Existem três formas principais de incluir suporte para expressões SQL em linguagens de programação:
  - **Comandos embebidos**: os comandos SQL são identificados por prefixos especiais que são processados por um pré-compilador do próprio SGBD que os substitui por código com chamadas a funções próprias do SGBD.
  - **Biblioteca de funções**: o SGBD disponibiliza um conjunto de funções próprias de comunicação (*API – Application Programming Interface*) que permitem ligar à BD, executar consultas, inserir, remover e alterar dados, ...
  - **Novas linguagens de programação**: linguagens desenhadas de base e que incorporam desde logo conceitos relacionados com o modelo e a linguagem de manipulação da BD.

Problemas de Integração

- *Impedance mismatch*
  - Como lidar com tipo de dados diferentes e/ou incompatíveis?
  - Como lidar com resultados (dos comandos SQL) que são conjuntos de valores?
  - O modelo de dados do SQL (relações de tuplos de valores) é muito diferente do modelo de dados das outras linguagens (inteiros, reais, caracteres, arrays, estruturas, apontadores, ...).
- Outros problemas
  - Como passar valores do programa para uma expressão SQL?
  - Como retornar o resultado de um comando SQL para as variáveis do programa?

SQL Embebido em C

- Declarar variáveis SQL.

```
EXEC SQL BEGIN DECLARE SECTION;
/* variáveis especiais de ligação com o SGBD */
int SQLCODE;
char SQLSTATE[6];
/* variáveis partilhadas */
int bi;
char nomep[50], nomef[50];
float salário;
EXEC SQL END DECLARE SECTION;
```

- As variáveis **SQLCODE** e **SQLSTATE** são variáveis especiais utilizadas pelo SGBD para comunicar erros e excepções ao programa.
- As restantes variáveis podem ser usadas em C e em expressões SQL. Quando utilizadas em expressões SQL devem ser precedidas de dois pontos (**:**).

SQL Embebido em C

- Sempre que um comando SQL é executado, o SGBD retorna um valor em **SQLCODE** e **SQLSTATE**.
  - **(SQLCODE == 0)** significa que o comando foi executado com sucesso.
  - **(SQLCODE > 0)** significa que não existem mais tuplos no resultado de um consulta.
  - **(SQLCODE < 0)** significa que ocorreu um erro.
  - **(SQLSTATE == '00000')** significa que não ocorreu nenhum erro ou excepção.
  - **(SQLSTATE == '02000')** significa que não existem mais tuplos no resultado de um consulta.
  - ...
- Os códigos de erro e excepção devolvidos em **SQLCODE** são específicos de cada SGBD, enquanto que os do **SQLSTATE** são supostamente standard para todas as plataformas e versões.

SQL Embebido em C

- Estabelecer uma ligação com o SGBD (várias ligações são possíveis mas apenas uma pode estar activa num determinado momento).

```
CONNECT TO nome_servidor AS nome_ligação
AUTHORIZATION nome_utilizador_e_password;
```

- Definir a ligação activa (ligação sobre a qual os próximos comandos SQL passam a ter efeito).

```
SET CONNECTION nome_ligação;
```

- Terminar uma ligação com o SGBD.

```
DISCONNECT nome_ligação;
```


SQL Embebido em C

- Executar um comando SQL.

```
bi = obter_BI();
EXEC SQL
   SELECT NomeP, NomeF INTO :nomep, :nomef
   FROM EMPREGADO
   WHERE NumBI = :bi;
if (SQLCODE == 0)
   printf("BI: %d Nome: %s %s\n", bi, nomep, nomef);
```

- A declaração **EXEC SQL** especifica os blocos de código que definem expressões SQL.
- A declaração **INTO** resolve o problema de como retornar o resultado de um comando SQL para as variáveis do programa, pois permite indicar as variáveis que devem conter o resultado. Os nomes das variáveis não precisam de ser iguais nem diferentes dos nomes dos atributos das tabelas.

Cursores

- Os cursores são necessários quando um comando SQL devolve mais do que um tuplo como resultado. Um cursor é como que um apontador que em cada momento referencia um único tuplo do resultado (designado como o tuplo corrente).
- Declarar um cursor.

```
EXEC SQL DECLARE nome_cursor CURSOR FOR consulta_sql;
```

- Executar a consulta associada a um cursor e obter o seu resultado. O cursor fica a referenciar a posição (virtual) anterior ao primeiro tuplo.

```
EXEC SQL OPEN CURSOR nome_cursor;
```

- Deslocar um cursor para o tuplo seguinte e copiar os valores desse tuplo para variáveis do programa.

```
EXEC SQL FETCH FROM nome_cursor INTO lista_variáveis;
```

- Terminar de utilizar um cursor.

```
EXEC SQL CLOSE CURSOR nome_cursor;
```


Cursores

- Rotina para obter o nome dos empregados do departamento de Produção.

```
EXEC SQL DECLARE emp_prod CURSOR FOR
   SELECT NomeP, NomeF
   FROM EMPREGADO, DEPARTAMENTO
   WHERE NumDep = Num AND Nome = "Produção";
EXEC SQL OPEN CURSOR emp_prod;
EXEC SQL FETCH FROM emp_prod INTO :nomep, :nomef;
while (SQLCODE == 0) {
   printf("%s %s\n", nomep, nomef);
   EXEC SQL FETCH FROM emp_prod INTO :nomep, :nomef;
}
EXEC SQL CLOSE CURSOR emp_prod;
```

Cursores

- Quando se pretende efectuar alterações sobre os tuplos referenciados por um cursor é necessário incluir na declaração do cursor a expressão **FOR UPDATE OF** seguido da lista de atributos a alterar.

```
EXEC SQL DECLARE nome_cursor CURSOR FOR consulta_sql
FOR UPDATE OF lista_atributos;
```

- Caso se pretenda efectuar remoções basta incluir a expressão **FOR UPDATE**.

```
EXEC SQL DECLARE nome_cursor CURSOR FOR consulta_sql
FOR UPDATE;
```

- A especificação do tuplo a alterar/remover é depois feita através da expressão **CURRENT OF** na condição **WHERE** do comando **UPDATE/DELETE** respectivo.

```
EXEC SQL [ UPDATE nome_tabela SET nome_atributo = valor
         | DELETE FROM nome_tabela]
WHERE CURRENT OF nome_cursor;
```


Cursores

- Rotina para aumentar o salário dos empregados do departamento de Produção.

```
EXEC SQL DECLARE emp_prod CURSOR FOR
   SELECT NumBI, Salário FROM EMPREGADO, DEPARTAMENTO
   WHERE NumDep = Num AND Nome = "Produção"
   FOR UPDATE OF Salário;
EXEC SQL OPEN CURSOR emp_prod;
EXEC SQL FETCH FROM emp_prod INTO :bi, :salário;
while (SQLCODE == 0) {
   float aumento = calcula_aumento(bi);
   EXEC SQL
      UPDATE EMPREGADO SET Salário = Salário + :aumento
      WHERE CURRENT OF emp_prod;
   EXEC SQL FETCH FROM emp_prod INTO :bi, :salário;
}
EXEC SQL CLOSE CURSOR emp_prod;
```

Cursores

- Quando se pretende deslocar o cursor de forma não sequencial sobre o conjunto de tuplos é necessário incluir na declaração do cursor a expressão **SCROLL**.

```
EXEC SQL DECLARE nome_cursor SCROLL CURSOR FOR consulta_sql;
```

- Os deslocamentos possíveis são:
  - **FETCH [NEXT]** – desloca o cursor para o tuplo seguinte (opção por defeito).
  - **FETCH PRIOR** – desloca o cursor para o tuplo anterior.
  - **FETCH FIRST** – desloca o cursor para o primeiro tuplo.
  - **FETCH LAST** – desloca o cursor para o último tuplo.
  - **FETCH ABSOLUTE N** – desloca o cursor para o tuplo na posição **N**.
  - **FETCH RELATIVE N** – desloca o cursor **N** posições a partir do tuplo corrente.

API C do MySQL

- O SGBD MySQL disponibiliza (entre outras) uma biblioteca para a linguagem C para acesso a bases de dados em MySQL. Essa biblioteca define uma API com as seguintes funcionalidades:
  - Funções para estabelecer e terminar ligações com o SGBD MySQL.
  - Funções para construir, enviar e processar consultas.
  - Funções para o tratamento de erros.
- Em relação ao SQL embebido, a utilização de uma API como forma de comunicação com o SGBD tem algumas vantagens e desvantagens:
  - É mais flexível pois permite aceder simultaneamente a várias BDs e em diferentes SGBDs.
  - É mais dinâmico pois as consultas não necessitam de estar previamente definidas.
  - É mais vulnerável a erros pois a verificação da sintaxe SQL só pode ser feita em tempo de execução.
  - Não necessita de pré-processamento, mas por outro lado necessita que a biblioteca do SGDB esteja instalada.

API C do MySQL

- Iniciar a estrutura de dados associada com uma ligação.
```
nome_ligação = mysql_init(...);
```
- Estabelecer uma ligação com o SGBD (várias ligações são possíveis).
```
mysql_real_connect(nome_ligação, ...);
```
- Terminar uma ligação com o SGBD.
```
mysql_close(nome_ligação);
```
- Enviar um comando SQL para o SGBD.
```
mysql_query(nome_ligação, comando_sql);
```
- Executar o comando enviado previamente e obter o seu resultado.
```
nome_resultado = mysql_store_result(nome_ligação);
```
- Obter o próximo tuplo de um resultado.
```
tuplo = mysql_fetch_row(nome_resultado);
```
- Libertar um resultado.
```
mysql_free_result(nome_resultado);
```


API C do MySQL

- Código-tipo de uma aplicação que utiliza a API C do MySQL.

```
#include <mysql.h>

int main() {
   MYSQL *ligação;
   ligação = mysql_init(...);
   mysql_real_connect(ligação, ...);
   while(...) {
      ... /* processar comandos SQL */
   }
   mysql_close(ligação);
}
```

API C do MySQL

- Código-tipo de uma aplicação que utiliza a API C do MySQL.

```
/* processar comandos SQL */
while(...) {
   MYSQL_RES *resultado;
   MYSQL_ROW tuplo;
   char *comando_sql;
   comando_sql = ...;  /* obter/construir um novo comando */
    /* os comandos SQL são criados dinamicamente em tempo */
    /* de execução e passados como strings à API do MySQL */
   mysql_query(ligação, comando_sql);
   resultado = mysql_store_result(ligação);
   while ((tuplo = mysql_fetch_row(resultado)) != NULL) {
      ... /* processar o tuplo corrente */
   }
   mysql_free_result(resultado);
}
```


API C do MySQL

- Código-tipo de uma aplicação que utiliza a API C do MySQL.

```
/* processar o tuplo corrente */
while ((tuplo = mysql_fetch_row(resultado)) != NULL) {
   /* tuplo é um apontador para um array de strings */
   /* onde estão os valores dos atributos do tuplo. */
   /* Para manipular valores numéricos é necessário */
   /* converter previamente as strings.             */
   int x = converte_para_inteiro(tuplo[0]);
   printf("%d %s\n", x, tuplo[1]);
}
```

Programação no SGBD

- Muitos SGBDs possuem linguagens de programação próprias que para além do SQL incluem suporte para variáveis, estruturas de controle (IF – WHILE – FOR), procedimentos, funções, etc.
- Este tipo de programação tem as suas vantagens:
  - Reduz a diferença entre o SQL e a linguagem de programação.
  - É guardada no SGBD o que permite que possa ser invocado por diferentes aplicações evitando assim a duplicação de código.
  - Executa no SGBD o que reduz custos de comunicação.
  - Pode ser utilizado directamente na definição de triggers.
- O SQL/PSM (SQL/Persistent Stored Modules) é uma extensão ao SQL que define um standard para a escrita de procedimentos e funções em SQL que juntamente com a utilização de estruturas de controle aumentam consideravelmente o poder expressivo do SQL.

SQL/PSM

- Criar um procedimento.

```
CREATE PROCEDURE nome(argumentos)
declaração_de_variáveis
corpo_do_procedimento;
```

- Criar uma função.

```
CREATE FUNCTION nome(argumentos)
RETURNS tipo_de_dados
declaração_de_variáveis
corpo_da_função;
```

- Invocar um procedimento ou função.

```
CALL nome(argumentos);
```

SQL/PSM

- Função para quantificar o tamanho de um departamento.

```
/* IN significa que o argumento é de input */
CREATE FUNCTION tamanho_departamento(IN numdep INT)
RETURNS VARCHAR[8]
/* Declaração de variáveis */
DECLARE total_emps INT;
/* Corpo da função */
SELECT COUNT(*) INTO total_emps
FROM EMPREGADO
WHERE NumDep = numdep;
IF total_emps > 100 THEN RETURN "Enorme"
ELSEIF total_emps > 50 THEN RETURN "Grande"
ELSEIF total_emps > 30 THEN RETURN "Médio"
ELSE RETURN "Pequeno"
ENDIF;
```


Transacções

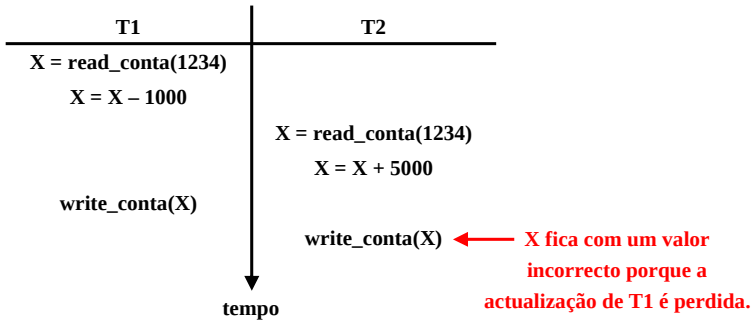
- Uma transacção é uma sequência atómica de operações sobre a base de dados.
  - Uma transacção ou é executada na totalidade ou não se realiza. Quando uma transacção é submetida para execução, o SGBD garante que ou todas as operações na transacção são executadas com sucesso e o seu efeito fica reflectido na BD ou a transacção não tem qualquer efeito na base de dados ou em outras transacções.
  - Uma transacção pode ser especificada interactivamente via SQL ou embebida na execução de um programa que acede ou modifica a base de dados.
- Sistemas de processamento de transacções são sistemas com bases de dados enormes e com centenas de utilizadores a aceder simultaneamente aos dados.
  - Estes sistemas requerem alta disponibilidade e rápido tempo de resposta para todos os utilizadores.
  - Exemplos destes sistemas são os bancos, os supermercados, as bolsas, os sistemas de reservas de avião, entre outros.

Transacções

- O conceito de transacção providencia um mecanismo para descrever unidades lógicas de processamento. Este conceito surge motivado por duas propriedades importantes de um SGBD:
  - **Acesso multi-utilizador**: para permitir concorrência no acesso aos dados é necessário ter mecanismos de controle que garantam a consistência e correcção nas actualizações sobre a base de dados.
  - **Protecção contra falhas**: se o sistema falha a meio do processamento de uma transacção a base de dados pode ficar num estado inválido. É necessário ter mecanismos de recuperação de falhas.

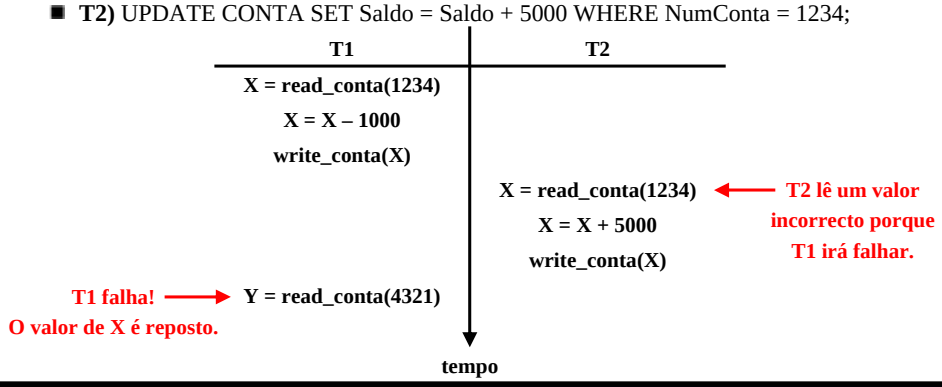
Problemas Relacionados com Concorrência

- **Perda de actualizações**
  - **T1)** UPDATE CONTA SET Saldo = Saldo – 1000 WHERE NumConta = 1234;
  - **T2)** UPDATE CONTA SET Saldo = Saldo + 5000 WHERE NumConta = 1234;



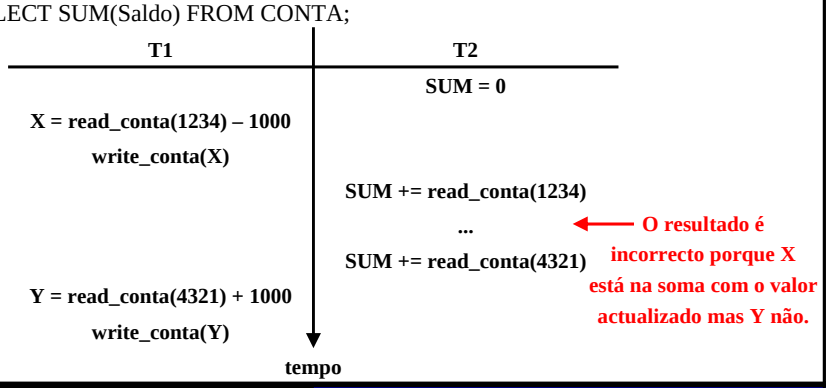
Problemas Relacionados com Concorrência

- **Actualização temporária (*dirty read*)**
  - **T1)** UPDATE CONTA SET Saldo = Saldo – 1000 WHERE NumConta = 1234;
    UPDATE CONTA SET Saldo = Saldo + 1000 WHERE NumConta = 4321;
  - **T2)** UPDATE CONTA SET Saldo = Saldo + 5000 WHERE NumConta = 1234;



Problemas Relacionados com Concorrência

- **Resumo incorrecto**
  - **T1)** UPDATE CONTA SET Saldo = Saldo – 1000 WHERE NumConta = 1234;
    UPDATE CONTA SET Saldo = Saldo + 1000 WHERE NumConta = 4321;
  - **T2)** SELECT SUM(Saldo) FROM CONTA;



Log de Transacções

- Tendo em vista a recuperação de falhas, os SGBDs mantêm um ficheiro, chamado de *log*, onde registam o percurso de execução das transacções.
- O ficheiro de *log* é mantido em disco (do qual deve existir um *backup*) de modo a ficar salvaguardado de outras falhas (excepto falha do próprio disco).
- As entradas registadas no ficheiro de *log* são do seguinte tipo (**`T`** é o identificador único gerado pelo próprio sistema para identificar uma dada transacção):
  - **`[start_transaction, T]`**: indica que foi iniciada a execução da transacção **`T`**.
  - **`[read_item, T, X]`**: indica que **`T`** leu o item **`X`**.
  - **`[write_item, T, X, OldVal, NewVal]`**: indica que **`T`** modificou o item **`X`** de **`OldVal`** para **`NewVal`**.
  - **`[commit, T]`**: indica que **`T`** terminou com sucesso e que as alterações feitas por **`T`** são válidas. Sempre que ocorre um *commit*, os blocos do ficheiro de *log* que estão a ser modificados em memória são escritos para disco.
  - **`[abort, T]`**: indica que **`T`** foi interrompido e não sucedeu.

Checkpoints

- Para simplificar o processo de recuperação após falha, o SGBD procede a *checkpoints* regulares, adicionando ao ficheiro de *log* a marca **`[checkpoint]`**.
- Assim todas as transacções que tenham uma entrada *commit* no ficheiro de *log* antes da última entrada *checkpoint* não necessitam de recuperação.
- A realização de um *checkpoint* envolve:
  - Suspender temporariamente as transacções em execução.
  - Forçar a escrita da memória para o disco de todos os blocos com modificações.
  - Escrever a marca **`[checkpoint]`** no ficheiro de *log* e forçar a escrita do ficheiro de *log* para disco.
  - Retomar a execução das transacções suspensas.

Transacções

- Propriedades desejáveis das transacções
  - **Atomicidade**: uma transacção é uma unidade de processamento atómica, ou é realizada na totalidade ou não se realiza.
  - **Consistência**: a correcta execução de uma transacção deve conduzir a base de dados de um estado consistente a outro.
  - **Isolamento**: a execução de uma transacção não deve interferir no resultado de transacções concorrentes. O isolamento pode ser conseguido através da sequencialização, isto é, permitindo-se intercalar a execução de operações de transacções desde que o resultado da sua execução seja equivalente ao da execução sequencial.
  - **Durabilidade ou persistência**: o resultado de uma transacção deve persistir na base de dados e não deve ser perdido por falha do sistema.

Transacções em SQL

- A implementação de transacções é conseguida por utilização das seguintes declarações:
  - **`EXEC SQL SET TRANSACTION`**: indica o início da execução de uma transacção.
  - **`EXEC SQL COMMIT`**: indica o fim com sucesso da transacção de modo a que as alterações feitas à base de dados pela transacção possam ser tornadas permanentes.
  - **`EXEC SQL ROLLBACK`**: indica o fim sem sucesso da transacção de modo a que as alterações feitas à base de dados pela transacção possam ser invalidadas.
- Na declaração **`EXEC SQL SET TRANSACTION`** podemos ainda indicar:
  - O tipo de acesso da transacção: **`READ WRITE`** (valor por defeito) ou **`READ ONLY`**.
  - O nível de isolamento: **`ISOLATION LEVEL`** que pode ser **`SERIALIZABLE`** (valor por defeito), **`READ COMMITED`** (não evita leituras diferentes do mesmo valor) ou **`READ UNCOMMITED`** (não evita *dirty-reads* nem leituras diferentes do mesmo valor).

Transacções em SQL

- Rotina para transferir dinheiro entre 2 contas.

```
EXEC SQL BEGIN DECLARE SECTION;
   int conta1, conta2, valor;
EXEC SQL END DECLARE SECTION;
obter_dados(conta1, conta2, valor);
EXEC SQL SET TRANSACTION;
EXEC SQL UPDATE CONTA SET Saldo = Saldo – :valor
         WHERE NumConta = :conta1;
if (SQLCODE != 0) EXEC SQL ROLLBACK;
else {
   EXEC SQL UPDATE CONTA SET Saldo = Saldo + :valor
         WHERE NumConta = :conta2;
   if (SQLCODE != 0) EXEC SQL ROLLBACK;
   else EXEC SQL COMMIT;
}
```


Segurança em Bases de Dados

- Um SGBD deve possuir mecanismos que permitam evitar a perca ou degradação dos seguintes objectivos de segurança:
  - **Perca de integridade**: é necessário proteger a BD contra modificações intencionais ou acidentais que corrompam a BD e levem a decisões erradas, incorrectas ou fraudulentas.
  - **Perca de disponibilidade**: é necessário garantir que a informação da BD não fica indisponível para quem tem o legítimo direito de a aceder e manipular.
  - **Perca de confidencialidade**: é necessário garantir que a informação da BD não é acessível por parte de utilizadores não autorizados.

Segurança em Bases de Dados

- O administrador da BD (DBA) é a entidade central na gestão de um SGBD. O DBA possui uma conta privilegiada que lhe permite:
  - Criar novas contas de utilizador que permitam o acesso ao SGBD.
  - Definir os privilégios associados a cada conta.
  - Atribuir níveis de segurança às diferentes contas. Por exemplo, para determinadas contas o SGBD pode estender o ficheiro *log* do sistema para guardar todas as operações realizadas a partir dessa conta por forma a que seja possível auditar esse conjunto de operações caso seja necessário (muito utilizado em BD de bancos).

Segurança em Bases de Dados

- Os privilégios associados às contas de um SGBD dividem-se em privilégios ao:
  - **Nível da conta**: privilégios que determinam se o utilizador pode criar tabelas (CREATE TABLE); criar visões (CREATE VIEW); alterar tabelas (ALTER); remover tabelas ou visões (DROP); inserir, remover ou alterar dados (MODIFY); ou obter dados da base de dados (SELECT). Estes privilégios aplicam-se à conta em geral e são independentes das relações (tabelas/visões) existentes.
  - **Nível das tabelas**: privilégios que determinam se o utilizador pode obter dados de uma tabela (SELECT); inserir, remover ou alterar dados de uma tabela (MODIFY ou INSERT/DELETE/UPDATE); ou referenciar uma tabela numa restrição de integridade (REFERENCES). Estes privilégios determinam o modo como o utilizador pode interagir com cada tabela (ou visão).
- Os privilégios ao nível da conta não fazem parte do SQL, a sua definição depende do SGBD utilizado.

Privilégios em SQL

- Obtenção de privilégios em SQL.
  - O utilizador que cria uma nova tabela (ou visão) fica com todos os privilégios sobre essa tabela podendo transmiti-los a terceiros.
  - Por transmissão via o comando GRANT.
- Transmitir privilégios sobre um conjunto de tabelas a um grupo de utilizadores.

  GRANT <PRIVILÉGIOS> **ON** <TABELAS>

  TO <UTILIZADORES> [**WITH GRANT OPTION**];
  - <PRIVILÉGIOS> é a lista dos privilégios a transmitir. **ALL PRIVILEGES** permite transmitir todos os privilégios.
  - <TABELAS> é a lista das tabelas a considerar.
  - <UTILIZADOR> é a lista das contas a considerar. **PUBLIC** permite indicar todas as contas.
  - A opção **WITH GRANT OPTION** permite que os utilizadores que recebem os privilégios possam também eles os transmitir a terceiros.

Privilégios em SQL

- Cancelar privilégios sobre um conjunto de tabelas a um grupo de utilizadores.

  REVOKE <PRIVILÉGIOS> **ON** <TABELAS>

  FROM <UTILIZADORES> [**CASCADE**];
  - A opção **CASCADE** cancela também os privilégios atribuídos em função dos privilégios agora cancelados.
- Cancelar o privilégio de transmitir privilégios.

  REVOKE GRANT OPTION FOR <PRIVILÉGIOS> **ON** <TABELAS>

  FROM <UTILIZADORES>;

Privilégios em SQL

- Considere 3 utilizadores A1, A2 e A3 e suponha que o utilizador A1 cria a tabela EMPREGADO.

- A1 transmite o privilégio de acesso a EMPREGADO a A2 e permite-lhe que propague esse privilégio a outros.

  GRANT SELECT ON EMPREGADO **TO** A2 **WITH GRANT OPTION**;

- A2 transmite o privilégio de acesso a EMPREGADO a A3.

  GRANT SELECT ON EMPREGADO **TO** A3;

- A1 cancela todos os privilégios de acesso a EMPREGADO.

  REVOKE SELECT ON EMPREGADO **FROM** A2 **CASCADE**;