GUIA DE LABORATÓRIO 2.2
CONSULTAS AVANÇADAS

OBJECTIVOS

- Utilizar o comando `SELECT` com cláusulas `JOIN` para definir consultas multitabela.
- Distinguir diversos tipos de junções entre tabelas
- Consultas Sumário, Funções Agregadas/Agregadoras, `ORDER BY`, `GROUP BY`, `DISTINCT`, etc.

INSTRUÇÕES

1a Parte – Consultas multitabela com junção interna (`INNER JOIN`)

1. Inicie o HeidiSQL da forma habitual.

2. Vamos listar todas as acções, indicando o id da acção, o nome do curso, o número da acção, datas inicial e final, e duração total da acção. Introduza:

```
SELECT Accao.ID,
       Curso.Nome,
       Accao.Numero,
       Accao.DataInicial,
       Accao.DataFinal,
       Curso.Duracao
FROM   Accao
JOIN   Curso
ON     Accao.IDCurso = Curso.ID;
```

*Uma cláusula **`JOIN`** (junção) combina colunas de uma mais tabelas utilizando valores comuns a ambas as tabelas. Tipicamente ligamos tabelas através de um **`JOIN`** assente na associação entre uma chave-estrangeira e a respectiva chave-primária na tabela associada. A junção em cima indicada é uma junção interna (**`INNER JOIN`**, sendo que a palavra-reservada **`INNER`** é opcional) uma vez que apenas une as linhas da tabela `Accao` às linhas da tabela `Curso` que tenham como `ID` o valor de `Accao.IDCurso`. Se existissem acções sem linhas correspondentes na tabela `Curso` (o que neste caso não é possível devido à integridade referencial), então essas linhas não seriam devolvidas numa junção interna.*

3. Agora vamos seleccionar os nomes de todos os estudantes, bem como, os respectivos cursos (os nomes e não os IDs) e durações:

```
SELECT Estudante.Nome, Curso.Nome, Duracao
FROM   Estudante
JOIN   Inscricao
ON     Inscricao.IDEstudante = Estudante.ID
JOIN   Accao
ON     Inscricao.IDAccao = Accao.ID
JOIN   Curso
ON     Accao.IDCurso = Curso.ID;
```

4. A consulta anterior também pode ser escrita do seguinte modo:

```
SELECT Estudante.Nome, Curso.Nome, Duracao
FROM   Estudante, Inscricao, Accao, Curso
WHERE  Inscricao.IDEstudante = Estudante.ID
       AND Inscricao.IDAccao = Accao.ID
       AND Accao.IDCurso = Curso.ID;
```

5. Altere a consulta anterior de modo a que as colunas `Nome` dos resultados indiquem a que nomes se referem.

6. Vamos redefinir a consulta do laboratório anterior que devolvia os alunos dos cursos relacionados com natureza de modo a incluir o nome desses cursos:

```
SELECT Estudante.Nome, Curso.Nome, Duracao
FROM   Estudante
JOIN   Inscricao
ON     Inscricao.IDEstudante = Estudante.ID
JOIN   Accao
ON     Inscricao.IDAccao = Accao.ID
JOIN   Curso
ON     Accao.IDCurso = Curso.ID
WHERE  Curso.Nome IN ('Biologia', 'Agricultura Aplicada');
```

7. Podemos criar uma vista reunindo informação de `Curso` e `Accao` (similar à primeira consulta deste laboratório) e uma outra para associar `Inscricao` e `Estudante`:

```
CREATE VIEW AccaoCurso AS
   SELECT Accao.ID AS IDAccao,
          Curso.Nome AS NomeCurso,
          Accao.Numero,
          Accao.DataInicial,
          Accao.DataFinal,
          Curso.Duracao,
          Curso.Tipo
   FROM   Accao
   JOIN   Curso
   ON     Accao.IDCurso = Curso.ID;
```

```
CREATE VIEW InscricaoEstudante AS
   SELECT Inscricao.ID AS IDInscricao,
          Inscricao.Classificacao,
          Inscricao.IDAccao,
          Inscricao.DataInscricao,
          Estudante.ID AS IDEstudante,
          Estudante.Nome AS NomeEstudante,
          Estudante.Apelido AS Apelido
   FROM   Inscricao
   JOIN   Estudante
   ON     Inscricao.IDEstudante = Estudante.ID
```

8. E agora a consulta com informação sobre os estudantes de cursos relacionados com natureza fica mais simples porque utiliza as vistas criadas:

```
SELECT NomeEstudante, NomeCurso, Duracao
FROM   AccaoCurso
JOIN   InscricaoEstudante
ON     AccaoCurso.IDAccao = InscricaoEstudante.IDAccao
WHERE  NomeCurso IN ('Biologia', 'Agricultura Aplicada');
```

9. Introduza o seguinte comando para obter os nomes, datas de inscrição e NIFs de todos os estudantes trabalhadores:

```
SELECT Nome, DataEstatuto, NIF
FROM   Estudante, EstudanteTrabalhador
WHERE  Estudante.ID = EstudanteTrabalhador.IDEstudante;
```

10. Acrescente à consulta anterior a data da inscricao e o nome do curso do estudante:

```
SELECT NomeEstudante  AS 'Nome estudante',
       NomeCurso      AS 'Curso',
       DataInscricao  AS 'Data de inscricao',
       NIF
FROM   AccaoCurso, InscricaoEstudante, EstudanteTrabalhador
WHERE  AccaoCurso.IDAccao = InscricaoEstudante.IDAccao AND
       InscricaoEstudante.IDEstudante = EstudanteTrabalhador.IDEstudante;
```

11. Podemos dar nomes mais curtos às tabelas envolvidas na consulta através da cláusula **AS** :

```
SELECT NomeEstudante  AS 'Nome estudante',
       NomeCurso      AS 'Curso',
       DataInscricao  AS 'Data de inscricao',
       NIF
FROM   AccaoCurso AS AC, InscricaoEstudante AS IE, EstudanteTrabalhador AS ET
WHERE  AC.IDAccao = IE.IDAccao AND
       IE.IDEstudante = ET.IDEstudante;
```

12. Introduza o seguinte comando para obter os nomes, datas de inscrição e NIFs de todos os estudantes, trabalhadores ou não trabalhadores:

```
SELECT NomeEstudante  AS 'Nome estudante',
       NomeCurso      AS 'Curso',
       DataInscricao  AS 'Data de inscricao',
       COALESCE(NIF, '<nao aplicavel>') AS NIF
FROM   InscricaoEstudante AS IE
JOIN   AccaoCurso AS AC
ON     IE.IDAccao = AC.IDAccao
LEFT OUTER JOIN EstudanteTrabalhador AS ET
ON     IE.IDEstudante = ET.IDEstudante;
```

2a Parte – ORDER BY, LIMIT

13. Suponha agora que pretende obter uma listagem de todos os formandos ordenados ascedentemente pela classificação. Podemos utilizar a cláusula `ORDER BY` para tal

```
SELECT NomeEstudante,
       Classificacao
FROM   InscricaoEstudante
ORDER BY Classificacao;   -- por omissão a ordem é ASC
```

> *A cláusula **`ORDER BY`** é utilizada para ordenar as linhas do resultado de uma consulta. É muito utilizada por si só ou em consultas sumário. A sua sintaxe geral é:*
>
> ```
> SELECT expressão
> FROM tabelas
> [WHERE critérios]
> ORDER BY expressão [ ASC | DESC ];
> ```
>
> *Por omissão, a ordenação é ascendente.*

14. Vários estudantes podem ter a mesma classificação. Podemos escolher um segundo critério de ordenação para estes:

```
SELECT NomeEstudante,
       Classificacao,
       DataInscricao
FROM   InscricaoEstudante
ORDER BY Classificacao DESC, NomeEstudante ASC;
```

15. E agora podemos seleccionar as cinco melhores notas com a cláusula `LIMIT` e, à direita, as cinco melhores notas seguintes

```
SELECT NomeEstudante,
       Classificacao,
       DataInscricao
FROM   InscricaoEstudante
ORDER BY Classificacao DESC,
         NomeEstudante ASC
LIMIT  5;   -- mesmo que LIMIT 0, 5
```

```
SELECT NomeEstudante,
       Classificacao,
       DataInscricao
FROM   InscricaoEstudante
ORDER BY Classificacao DESC,
         NomeEstudante ASC
LIMIT  5, 5;
```

16. Por vezes necessitamos de ordenar as linhas para aplicar correctamente o `LIMIT`, mas depois queremos outra ordem para apresentação. Considere a seguinte consulta:

```
SELECT NomeEstudante,
       Classificacao,
FROM   InscricaoEstudante
ORDER BY Classificacao DESC
```

```
LIMIT 0, 7
```

17. Suponha que pretende exibir as linhas resultantes da consulta anterior por ordem ascendente. Vamos executar uma sub-consulta

```
SELECT NomeEstudante,
       Classificacao
FROM   (SELECT NomeEstudante,
               Classificacao
        FROM InscricaoEstudante
        ORDER BY Classificacao DESC
        LIMIT 7) AS M           -- sub-queries devem receber um alias ('M' neste caso)
ORDER BY Classificacao ASC;
```

18. Podemos especificar uma ordenação diferente com a função `FIELD`. Por exemplo, suponha que pretendia obter listar as inscrições por esta ordem do atributo `Estado`: primeiro, as com o estado `'activa'`, depois, as com estado `'suspensa'`, e, em último lugar, as inscrições `'concluida'`s:

```
SELECT IDEstudante,
       IDAccao,
       Estado
FROM   Inscricao
ORDER BY FIELD(Estado, 'activa', 'suspensa', 'concluida');
```

3a Parte – Consultas Sumário e Funções Agregadoras/Agregadas

> *Uma consulta sumário é uma consulta que resume uma ou mais linhas através de uma cláusula especial (eg, **GROUP BY** - ver à frente) e/ou de uma função agregada/agregadora. Através de uma consulta sumário podemos obter informação como, por exemplo, o valor mais alto de uma coluna, quantas linhas verificam determinado critério, a média de uma determinada coluna, etc.*

19. Vamos obter a classificação mais alta, a média e a mais baixa dos estudantes:

```
SELECT MAX(Classificacao), MIN(Classificacao), AVG(Classificacao)
FROM   InscricaoEstudante;
```

20. Para obter o número total de `Estudantes` podemos fazer:

```
SELECT COUNT(*)
FROM Estudante;
```

> *No contexto de um **SELECT**, a função agregada **COUNT(expr)** devolve uma contagem do número de linhas para as quais as expressão expr não é **NULL**. Na maioria dos casos utilizamos **COUNT(*)** e aí todas as linhas são contadas.*

21. Ou, caso queiramos saber, quantos obtiveram nota superior a 14:

```
SELECT COUNT(*)
FROM   InscricaoEstudante
WHERE  Classificacao >= 14;
```

22. A cláusula `DISTINCT` permit eliminar duplicados. Por exemplo, para obter as diferentes durações dos cursos temos a consulta na coluna da esquerda. As colunas podem ser combinadas. Na consulta à direita devolvemos pares únicos de cidade e código postal (admitindo que um código postal pode abranger mais do que uma cidade):

```
SELECT DISTINCT Duracao
FROM  Curso;
```

```
SELECT DISTINCT
       Cidade, CodigoPostal
FROM   Estudante;
```

23. Com a cláusula `GROUP BY` podemos agrupar linhas, normalmente para aplicar funções agregadas. Por exemplo, suponhamos que queremos obter uma contagem das acções (agrupadas) por `Curso`.

```
SELECT NomeCurso,
       Count(*) AS NumAccoes
FROM   AccaoCurso
GROUP BY NomeCurso;
```

*A cláusula **`GROUP BY`** é utilizada em conjugação com funções agregadas para agrupar as linhas do resultado pelo valor de uma ou mais colunas. A sua sintaxe geral é:*

```
SELECT coluna1, coluna2, ..., aggregate_function(coluna_i)
FROM tabela
WHERE critérios
GROUP BY coluna1, coluna2, ...;
```

*Um **`GROUP BY`** é semelhante a um **`DISTINCT`**. Na verdade, um **`DISTINCT`** é um **`GROUP BY`** sem ordenação dos resultados. Se adicionarmos um **`ORDER BY`** a um **`DISTINCT`** obtemos um **`GROUP BY`**.*

24. Ou, suponhamos que queremos a média das classificações por acção:

```
SELECT IDAccao,
       AVG(Classificacao) AS Média
FROM   InscricaoEstudante
GROUP BY IDAccao;
```

25. Ou a média das classificações por curso juntamente com o número de inscritos no curso:

```
SELECT C.ID, C.Nome, COUNT(I.IDEstudante), AVG(I.Classificacao)
FROM Inscricao AS I
JOIN Accao     AS A
ON   I.IDAccao = A.ID
JOIN Curso     AS C
ON   A.IDCurso = C.ID
GROUP BY C.ID
```

EXERCÍCIOS

1. Indique uma consulta para obter o nome do estudante cuja classificação é a mais alta.

2. Qual a finalidade da cláusula `HAVING`? Quais as diferenças para `WHERE`?

3. Indique uma consulta para obter:

3.1 Quantos estudantes inscritos em Biologia?

3.2 Listar o numero da acção bem como a duração de todas as ações de Contabilidade.

3.3 A média das notas dos estudantes de Agricultura.

3.4 O nome e a designação do curso dos 3 primeiros estudantes com média (ie, a classificação) mais alta?

3.5 O nome e a designação do curso de todos os estudante com média (ie, a classificação) mais alta?

4. Investigue a instrução `SELECT INTO` e o que são variáveis definidas pelo utilizador (com '@') e, de seguida, utilize ambos os conceitos para obter qual nome, data de nascimento e morada do estudante mais novo.