SQL — A Linguagem de Consulta

Exemplos de Tabelas

- Usaremos estas instâncias das relações “Pilotos” e “Reservas”.
  - Que “regra do negócio” seria diferente se a chave da relação Reservas contivesse só os atributos num_p e num_a?

R1

num_p	num_a	data
22	101	10/10/98
58	103	11/12/98

P1

num_p	nome_p	nível	idade
22	Joca	7	45
31	Laerte	8	55
58	Rafa	10	35

P2

num_p	nome_p	nível	idade
28	Zeca	9	35
31	Laerte	8	55
44	Giba	5	35
58	Rafa	10	35

Consulta SQL Básica

SELECT [DISTINCT] *lista-de-atribs*
FROM *lista-de-relações*
WHERE *condição*

- *lista-de-relações*: Uma lista de nomes de tabelas ou visões (possivelmente com uma variável tupla depois de cada nome).
- *lista-de-atribs*: Uma lista de nomes de colunas das tabelas ou visões na *lista-de-relações*.
- *condição*: comparações (Atrib *op* const ou Atrib1 *op* Atrib2, onde *op* é um de <, >, =, <=, >=, <>) combinadas com AND, OR ou NOT.
- DISTINCT: eliminar linhas repetidas.

Estratégia de Avaliação Conceitual

- A semântica de uma consulta SQL é definida pela seguinte estratégia de avaliação conceitual:
  - Compute o produto cartesiano da *lista-de-relações*.
  - Descarte as tuplas que não satisfazem a *condição*.
  - Remova colunas não listadas em *lista-de-atribs*.
  - Se houver DISTINCT, elimine repetições de linhas.
- Esta estratégia é provavelmente a maneira menos eficiente de se executar uma consulta! Um otimizador encontrará métodos mais eficientes que produzem os mesmos resultados.

Exemplo de Avaliação Conceitual

Nomes dos pilotos que reservaram o avião nº 103:

```
SELECT  P.nome_p
FROM    Pilotos P, Reservas R
WHERE  P.num_p = R.num_p  AND  R.num_a = 103
```

$$\Pi_{\text{Pilotos.nome\_p}}(\sigma_{\substack{\text{Pilotos.num\_p = Reservas.num\_p}\\ \text{AND}\\ \text{num\_a = 103}}}(\text{Pilotos} \times \text{Reservas}))$$

(num_p)	nome_p	nível	idade	(num_p)	num_a	data
22	Joca	7	45	22	101	10/10/98
22	Joca	7	45	58	103	11/12/98
31	Laerte	8	55	22	101	10/10/98
31	Laerte	8	55	58	103	11/12/98
58	Rafa	10	35	22	101	10/10/98
58	Rafa	10	35	58	103	11/12/98

Uma Nota Sobre Variáveis Tupla

- São de fato necessárias só no caso da mesma tabela aparecer mais de uma vez na cláusula FROM. A consulta anterior pode ser escrita assim:

```
SELECT  P.nome_p
FROM    Pilotos P, Reservas R
WHERE  P.num_p = R.num_p  AND  num_a = 103
```

Ou assim:

```
SELECT  nome_p
FROM    Pilotos, Reservas
WHERE  Pilotos.num_p = Reservas.num_p
    AND  num_a = 103
```

É considerado "bom estilo", no entanto, usar sempre variáveis tupla!

Números dos pilotos que reservaram algum avião
(pelo menos um)

- Vamos supor que existem várias tabelas com informações sobre pilotos (Pilotos, Pilotos2, etc.) e que na tabela Reservas são registradas as reservas feitas por pilotos das diferentes tabelas.
- Só estamos interessados nos pilotos que estão relacionados numa dessas tabelas (a tabela Pilotos).

```
SELECT  P.num_p
FROM    Pilotos P, Reservas R
WHERE   P.num_p = R.num_p
```

- Se estivéssemos interessados nos números de todos os pilotos (das várias tabelas) que reservaram algum avião, a consulta seria mais simples. (Como?)

Números dos pilotos que reservaram algum avião
(pelo menos um)

```
SELECT  P.num_p
FROM    Pilotos P, Reservas R
WHERE   P.num_p = R.num_p
```

- Faria alguma diferença especificar DISTINCT nesta consulta?
- Qual o efeito de se trocar P.num_p por P.nome_p na cláusula SELECT? Faria diferença especificar DISTINCT nesse caso?
- Para obter todos os atributos desses pilotos e de suas reservas, troque a primeira linha por

```
SELECT *   <-- todos os atributos
```

Expressões e Strings

```
SELECT  P.nivel, nivel1 = P.nivel - 5, 2 * P.nivel AS nivel2
FROM    Pilotos P
WHERE   P.nome_p LIKE 'B_%B'
```

- Ilustra o uso de expressões aritméticas e de strings com "curingas": *Encontrar triplas (nível do piloto e mais dois campos definidos por expressões) para pilotos cujos nomes começam e terminam com B e contém pelo menos três caracteres.*
- AS e = são duas maneiras de nomear colunas do resultado.
- LIKE é usado para comparações envolvendo curingas: '_' significa qualquer caracter e '%' significa 0 ou mais caracteres arbitrários.

Números dos pilotos que reservaram um Cessna ou um Learjet

```
SELECT P.num_p
FROM   Pilotos P, Avioes A, Reservas R
WHERE P.num_p = R.num_p AND A.num_a = R.num_a
    AND (A.tipo = 'Cessna' OR A.tipo = 'Learjet')
```

- O que obtemos trocando este OR por AND?

```
SELECT P.num_p
FROM   Pilotos P, Avioes A, Reservas R
WHERE P.num_p = R.num_p AND A.num_a = R.num_a
    AND A.tipo = 'Cessna'
UNION
SELECT P.num_p
FROM   Pilotos P, Avioes A, Reservas R
WHERE P.num_p = R.num_p AND A.num_a = R.num_a
    AND A.tipo = 'Learjet'
```

- UNION: união de dois conjuntos de tuplas compatíveis para união.

Números dos pilotos que reservaram um Cessna e um Learjet

```
SELECT P.num_p
FROM    Pilotos P, Avioes A1, Reservas R1,
        Avioes A2, Reservas R2
WHERE P.num_p = R1.num_p AND A1.num_a = R1.num_a
    AND P.num_p = R2.num_p AND A2.num_a = R2.num_a
    AND (A1.tipo = 'Cessna' AND A2.tipo = 'Learjet')
```

Mais complicado que o caso do "ou"!

```
SELECT P.num_p          <-- Chave!
FROM    Pilotos P, Avioes A, Reservas R
WHERE P.num_p = R.num_p AND A.num_a = R.num_a
    AND A.tipo = 'Cessna'
INTERSECT
SELECT P.num_p
FROM    Pilotos P, Avioes A, Reservas R
WHERE P.num_p = R.num_p AND A.num_a = R.num_a
    AND A.tipo = 'Learjet'
```

- INTERSECT: intersecção de dois conjuntos de tuplas compatíveis para união.

UNION, INTERSECT, EXCEPT

- Nos exemplos anteriores, contraste a simetria das consultas que usam UNION e INTERSECT com a diferença entre as outras versões das mesmas consultas.
- Também disponível: EXCEPT, que computa a diferença de conjuntos de tuplas.
  - No exemplo anterior, qual o efeito de se trocar INTERSECT por EXCEPT?
- INTERSECT e EXCEPT estão no padrão SQL-92, mas ainda não são suportados por todos os sistemas.
  - Alguns sistemas suportam MINUS em vez de EXCEPT.

Aninhamento de Consultas

Nomes dos pilotos que reservaram o avião nº 103:

```
SELECT  P.nome_p
FROM    Pilotos P
WHERE  P.num_p IN (SELECT R.num_p
                   FROM    Reservas R
                   WHERE R.num_a = 103)
```

- A cláusula WHERE pode conter uma sub-consulta! (Cláusulas FROM e HAVING também podem.)
- Para achar os pilotos que não reservaram o avião nº 103, use NOT IN.
- Semântica: Para cada tupla de Pilotos, verifique se a condição é satisfeita computando a sub-consulta.

Aninhamento de Consultas Correlacionadas

Nomes dos pilotos que reservaram o avião nº 103:

```
SELECT  P.nome_p
FROM Pilotos P
WHERE  EXISTS (SELECT *
               FROM Reservas R
               WHERE R.num_p = P.num_p
                  AND R.num_a = 103)
```

- O operador EXISTS verifica se um conjunto de tuplas é não-vazio.
- Este exemplo ilustra por que, em geral, a sub-consulta precisa ser recomputada para cada tupla de Pilotos.

Mais Operações Com Conjuntos

- Já vimos IN e EXISTS. Há também NOT IN e NOT EXISTS.
- Também disponíveis: *op* ANY e *op* ALL (onde *op* pode ser >, <, =, >=, <=, ou <>).
- Pilotos mais velhos que algum piloto chamado Zé:

```
SELECT  *
FROM    Pilotos P
WHERE   P.idade > ANY (SELECT P2.idade
                       FROM   Pilotos P2
                       WHERE P2.nome_p = 'Zé')
```


Usando IN em vez de INTERSECT

Números dos pilotos que reservaram um Cessna e um Learjet:

```
SELECT P.num_p
FROM   Pilotos P, Avioes A, Reservas R
WHERE P.num_p = R.num_p
    AND A.num_a = R.num_a AND A.tipo = 'Cessna'
    AND P.num_p IN (SELECT P2.num_p
                    FROM   Pilotos P2, Avioes A2, Reservas R2
                    WHERE P2.num_p = R2.num_p
                        AND A2.num_a = R2.num_a
                        AND A2.tipo = 'Learjet')
```

- De modo análogo, consultas com EXCEPT podem ser reescritas usando NOT IN.
- Para achar os nomes (em vez dos números) desses pilotos, basta trocar P.num_p por P.nome_p no primeiro SELECT.
  - Como fazer essa mudança na consulta com INTERSECT?

Divisão em SQL

Nomes dos pilotos que reservaram todos os aviões:

❶ Usando EXCEPT

```
SELECT P.nome_p
FROM   Pilotos P
WHERE NOT EXISTS
        ( (SELECT A.num_a
           FROM   Avioes A)
          EXCEPT
          (SELECT R.num_a
           FROM Reservas R
           WHERE R.num_p = P.num_p))
```

❷ Sem EXCEPT

```
SELECT P.nome_p
FROM   Pilotos P
WHERE NOT EXISTS (SELECT A.num_a
                  FROM   Avioes A
                  WHERE NOT EXISTS (SELECT R.num_a
                                    FROM   Reservas R
                                    WHERE R.num_a = A.num_a
                                    AND R.num_p = P.num_p))
```

Pilotos P tais que…

não há avião A…

sem uma reserva de A para P

Funções de Agregação

- Extensão significativa da álgebra relacional.

COUNT (*)
COUNT ([DISTINCT] *atrib*)
SUM ([DISTINCT] *atrib*)
AVG ([DISTINCT] *atrib*)
MAX (*atrib*)
MIN (*atrib*)

```
SELECT COUNT(DISTINCT P.nivel)
FROM Pilotos P
WHERE P.nome_p = 'Jorge'
```

```
SELECT AVG (P.idade)
FROM Pilotos P
WHERE P.nivel = 10
```

```
SELECT AVG (DISTINCT P.idade)
FROM Pilotos P
WHERE P.nivel = 10
```

```
SELECT COUNT (*)
FROM Pilotos P
```

```
SELECT P.nome
FROM Pilotos P
WHERE P.nivel = (SELECT MAX (P2.nivel)
                 FROM Pilotos P2)
```

Nome e idade do(s) piloto(s) mais velho(s)

- A primeira consulta é inválida! (Veremos a razão daqui a pouco, quando discutirmos GROUP BY.)
- A terceira consulta é equivalente à segunda e é permitida pelo padrão SQL-92, mas não é suportada por alguns sistemas.

```
SELECT P.nome_p, MAX (P.idade)
FROM Pilotos P

SELECT P.nome_p, P.idade
FROM Pilotos P
WHERE P.idade =
          (SELECT MAX (P2.idade)
           FROM Pilotos P2)

SELECT P.nome_p, P.idade
FROM Pilotos P
WHERE (SELECT MAX (P2.idade)
           FROM Pilotos P2)
          = P.idade
```


GROUP BY e HAVING

- Até agora aplicamos funções de agregação a todas as tuplas que satisfazem a condição no WHERE. Algumas vezes é desejável aplicá-las a vários grupos de tuplas.
- Exemplo: *Para cada nível, achar a idade do piloto mais jovem nesse nível.*
  - No caso geral, podemos não saber quantos níveis existem e quais os seus valores!
  - Supondo que existem 10 níveis numerados de 1 a 10, podemos escrever 10 consultas assim (!):

Para *i* = 1, 2, …, 10:

```
SELECT MIN (P.idade)
FROM Pilotos P
WHERE P.nivel = i
```

Consultas Com GROUP BY e HAVING

SELECT [DISTINCT] *lista-de-atribs-ou-agregações*
FROM *lista-de relações*
WHERE *condição*
GROUP BY *lista-de-atribs-para-agrupamento*
HAVING *condição-para-grupos*

A *lista-de-atribs-ou-agregações* contém:

❶ Nomes de atributos

❷ Termos com funções de agregação (ex: MIN (P.idade)).

Os nomes de atributos (❶) precisam estar todos contidos na *lista-de-atribs-para-agrupamento*. Intuitivamente, cada tupla do resultado corresponde a um grupo, e esses atributos precisam ter o mesmo valor sobre todo o grupo.

Avaliação Conceitual

- Efetue o produto cartesiano da *lista-de-relações*, elimine as tuplas que não satisfazem à *condição*, remova as colunas "desnecessárias" e particione as tuplas restantes em grupos, de acordo com a *lista-de-atribs-para-agrupamento*. (Um grupo é um conjunto de tuplas que coincidem sobre todos os atributos dessa lista.)
- Aplique a *condição-para-grupos* e elimine os grupos que não atenderem esse requisito. Toda expressão que aparecer nessa condição deverá ter um valor único por grupo!
  - Precisa estar na *lista-de-atribs-para-agrupamento* todo atributo que aparecer na *condição-para-grupos* e não for argumento para uma função de agregação. (SQL não explora a semântica de chaves aqui!)
- Para cada grupo "sobrevivente", gere uma tupla resultado.

Ache a idade do mais jovem dos pilotos com idade ≥ 18, para cada nível com pelo menos dois desses pilotos

```
SELECT P.nivel, MIN (P.idade)
FROM Pilotos P
WHERE P.idade >= 18
GROUP BY P.nivel
HAVING COUNT (*) > 1
```

- Só P.nivel e P.idade aparecem nas cláusulas SELECT, GROUP BY ou HAVING; os demais atributos são 'desnecessários'.
- A segunda coluna do resultado não tem nome. (Use AS para nomeá-la.)

num_p	nome_p	nível	idade
22	Joca	7	45
31	Laerte	8	55
71	Zildo	10	16
64	Henry	7	35
29	Bronco	1	33
58	Rafa	10	35

nível	idade
1	33
7	45
7	35
8	55
10	35

Nível	
7	35

Tabela resultado

Quantas reservas existem para cada avião tipo Learjet?

```
SELECT A.num_a, COUNT (*) AS reservas
FROM Pilotos P, Avioes A, Reservas R
WHERE P.num_p = R.num_p
    AND R.num_a = A.num_a AND A.tipo = 'Learjet'
GROUP BY A.num_a
```

- Formando grupos sobre a junção de tres relações.
- O que obtemos retirando A.tipo = 'Learjet' da cláusula WHERE e adicionando uma cláusula HAVING com essa condição?
- E se retirarmos Pilotos P e a condição sobre P.num_p?

Ache a idade do mais jovem dos pilotos com idade ≥ 18, para cada nível com pelo menos 2 pilotos (de ∀ idade)

```
SELECT P.nivel, MIN (P.idade)
FROM Pilotos P
WHERE P.idade >= 18
GROUP BY P.nivel
HAVING 1 < (SELECT COUNT (*)
            FROM Pilotos P2
            WHERE P.nivel = P2.nivel)
```

- Mostra uma cláusula HAVING contendo uma sub-consulta.
- Compare com a consulta que levava em conta somente níveis com pelo menos dois pilotos maiores de 18!
- E se a cláusula HAVING for trocada por:

  HAVING COUNT (*) > 1

Ache os níveis para os quais a média das idades dos pilotos é a mais baixa

- Funções de agregação não podem ser aninhadas!
  Solução errada:

```
SELECT P.nivel
FROM Pilotos P
WHERE P.idade = (SELECT MIN (AVG( P2.idade) ) )
                FROM Pilotos P2)
```

- Solução correta (em SQL-92):

```
SELECT Temp.nivel, Temp.idade_media
FROM (SELECT P.nivel, AVG (P.idade) AS idade_media
      FROM Pilotos P
      GROUP BY P.nivel) AS Temp
WHERE Temp.idade_media = (SELECT MIN (Temp.idade_media)
                         FROM Temp)
```

Valores Nulos (ou Vazios)

- Os atributos de uma tupla são às vezes desconhecidos (ex.: um nível que ainda não foi determinado) ou não aplicáveis (ex.: nome do conjuge, para uma pessoa solteira).
- Para tais situações, SQL tem um valor especial do atributo, o valor nulo (*null*).

☞ Null não é zero, é vazio!

Valores nulos complicam muitas questões:

- Operadores especiais são necessários para testar se um valor é nulo ou não.
- A condição *nivel > 8* é verdadeira ou falsa se o *nivel* for nulo? E condições contendo AND, OR e NOT?
- É necessária uma lógica a três valores (verdade, falso e desconhecido).
- O significado de construções com *nulls* tem que ser especificado cuidadosamente. (Por exemplo: a cláusula WHERE elimina as linhas para as quais a condição não é verdadeira.)
- Novos operadores de junção (*outer joins*) necessários.

Problema: SQL não é uma Linguagem de Programação!

- Não é computacionalmente completa.
  - Não tem comandos de controle de fluxo como *if… else*, *while*, *for*, etc.
  - Como programar aplicações de bancos de dados?
- Solução 1: estender SQL.
  - Fornecedores de SGBDs oferecem extensões computacionalmente completas (e proprietárias) de SQL:
    - PL/SQL (Oracle)
    - Transact-SQL (Sybase)
    - Access Basic (Microsoft Access)

Problema: SQL não é uma Linguagem de Programação!

- Solução 2: interfacear uma linguagem de programação com SQL. Dois tipos de interface são usados:
  - API de comandos embutidos
    - Comandos SQL são entremeados com comandos de uma linguagem de programação hospedeira (C ou COBOL, por exemplo).
    - Requer um pré-compilador que trata o SQL 'embutido' na linguagem hospedeira.
  - API de chamadas de função
    - Menos amigável.
    - Requer apenas uma biblioteca que implementa a API.

SQL Embutido (*Embedded SQL*)

- Comandos SQL são colocados diretamente no texto de um programa na linguagem hospedeira (C ou COBOL, por exemplo).
  - Comandos SQL podem se referir a variáveis da linguagem hospedeira (incluindo variáveis especiais para retorno de status).
  - Deve ser incluído um comando especial para conexão ao BD desejado.
- Em SQL, relações são (multi-)conjuntos de registros (tuplas), sem limite no número de registros. Tal estrutura de dados não existe em C.
  - SQL lida com esse problema oferecendo cursores.

Cursores

- Pode-se declarar um cursor sobre uma relação ou sobre uma consulta (cujo resultado é uma relação).
- Pode-se abrir um cursor, depois repetidamente buscar tuplas (*fetch*) e mover o cursor, até que todas as tuplas tenham sido obtidas.
  - A cláusula ORDER BY pode ser usada para controlar a ordem de obtenção das tuplas.
    - Os atributos na cláusula ORDER BY precisam aparecer também na cláusula SELECT.
- Pode-se também modificar ou remover a tupla apontada por um cursor.

Cursor que obtém, em ordem alfabética, os nomes dos pilotos que reservaram um avião tipo Cessna

```
EXEC SQL DECLARE info_p CURSOR FOR
        SELECT P.nome_p
        FROM Pilotos P, Avioes A, Reservas R
        WHERE P.num_p = R.num_p
            AND R.num_a = A.num_a AND A.tipo = 'Cessna'
        ORDER BY P.nome_p
```

- Note que não podemos trocar o atributo P.nome_p na cláusula ORDER BY por outro atributo — P.num_p, por exemplo. (Por quê?)
- Podemos adicionar P.num_p à cláusula SELECT e trocar P.nome_p por P.num_p na cláusula ORDER BY?

Embutindo SQL em C: Um Exemplo

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_nome_p[20]; short c_min_nivel; int c_idade;
EXEC SQL END DECLARE SECTION
c_min_nivel = ... ;
EXEC SQL DECLARE info_p CURSOR FOR
    SELECT P.nome_p, P.idade FROM Pilotos P
    WHERE P.nivel > :c_min_nivel
    ORDER BY P.nome_p;
do {
    EXEC SQL FETCH info_p INTO :c_nome_p, :c_idade;
    printf("%s tem idade %d\n", c_nome_p, c_idade);
} while (strcmp(SQLSTATE, "02000"));
EXEC SQL CLOSE info_p;
```

SQL: Conclusões

- Fator importante para a rápida aceitação do modelo relacional.
- Relacionalmente completa: Toda consulta que pode ser expressa na álgebra relacional pode também ser expressa em SQL (muitas vezes de modo mais natural).
- Tem poder expressivo significativamente maior que a álgebra relacional.
- Em geral há muitas maneiras de se escrever uma consulta. O otimizador de consultas deve escolher o plano de avaliação mais eficiente.
  - Na prática, usuários preocupados com eficiência obtém melhores resultados conhecendo detalhes do otimizador.