Structured Query Language - SQL

- Tópicos:
  - Linguagem de definição de dados
  - Estrutura básica de perguntas em SQL
  - Operações com conjuntos
  - Funções de agregação
  - Junções
  - Valores nulos
  - Subconsultas
  - Vistas
  - Modificações de bases de dados
- Bibliografia:
  - Capítulo 3 e Secções 4.1, 4.2, 4.4 e 4.5 do livro recomendado

Estrutura Básica

- SQL é baseado em operações de conjuntos e de álgebra relacional com algumas modificações e extensões
- Uma consulta SQL básica tem a forma:

  select A_1, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P

  - A_i*s* representam atributos
  - r_i*s* representam relações
  - P é um predicado.
- A consulta é equivalente à expressão de álgebra relacional (versão multi-conjunto):

$$\Pi_{A1,\ A2,\ ...,\ An}(\sigma_P\ (r_1\ \ x\ r_2\ \ x\ \ ...\ \ x\ \ r_m))$$

Funções de Agregação

- Estas funções aplicam-se a multiconjuntos de valores de uma coluna de uma relação, devolvendo um valor

avg: valor médio
min: valor mínimo
max: valor máximo
sum: soma dos valores
count: número de valores

Funções de Agregação (cont.)

- Determinar o saldo médio das contas na agência de Perryridge.

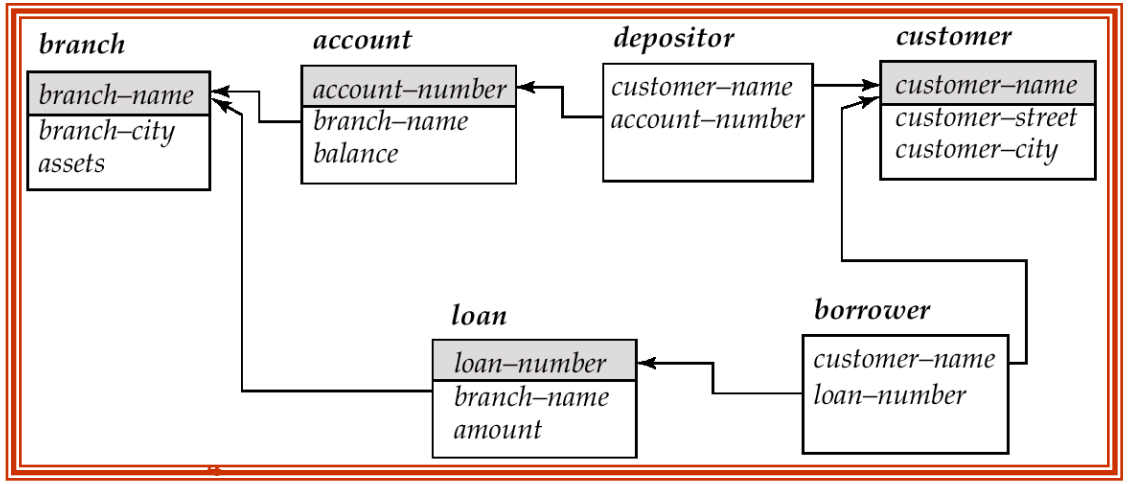
  select avg *(balance)*
  from *account*
  where *branch_name* = 'Perryridge'

- Calcular o número de clientes.

  select count (*)
  from *customer*

- Encontrar o número de depositantes do banco.

  select count (distinct *customer_name)*
  from *depositor*



Funções de agregação – Group By

- Listar o número de depositantes por agência.

  select *branch_name,* **count (distinct** *customer_name)*
  from *depositor, account*
  where *depositor.account_number = account.account_number*
  group by *branch_name*

  Nota: Atributos na cláusula **select** fora de funções de agregação têm de aparecer na lista **group by**

  Nota: Se aparecer mais do que um atributo em **group by**, então cada grupo é formado pelos tuplo com a mesma combinação de valores *em todos* esses os atributos

Funções de Agregação – Cláusula Having

- Listar os nomes de todas as agências cujo valor médio dos saldos das contas é superior a €1,200.

select *branch_name,* **avg** *(balance)*
from *account*
group by *branch_name*
having avg *(balance)* > 1200

Funções de Agregação – Cláusula Having

- Predicados na cláusula **having** são aplicados depois da formação dos grupos, enquanto que os predicados na cláusula **where** são aplicados antes da formação dos grupos.
- Encontrar a média dos saldos das contas dos clientes que vivem em Harrison e que têm pelo menos três contas.

```
select d.customer_name, avg (balance)
from depositor as d, account as a, customer as c
where d.account_number = a.account_number and
        d.customer_name = c.customer_name and
        c.city = 'Harrison'
group by d.customer_name
having count (distinct d.account_number) >= 3
```

Funções de agregação e álgebra relacional

- Um comando SQL genérico:

select $Fagr_1$, … $Fagr_n$
from R_1, …, R_m
where $Cond_W$
group by A_1, …, A_g
having $Cond_H$

- corresponde à expressão:

$$\Pi_{Fagr1, \ldots Fagrn} (\sigma_{Cond_H}(_{A1, \ldots, Ag}\mathcal{G}_{Fagr1, \ldots Fagrn}(\sigma_{Cond_W} (R_1 \times \ldots \times R_m))))$$

Funções de Agregação no Oracle

- AVG
- COLLECT
- CORR
- CORR_*
- COUNT
- COVAR_POP
- COVAR_SAMP
- CUME_DIST
- DENSE_RANK
- FIRST
- GROUP_ID
- GROUPING
- GROUPING_ID
- LAST
- MAX
- MEDIAN
- MIN
- PERCENTILE_CONT
- PERCENTILE_DISC
- PERCENT_RANK
- RANK
- REGR_ (Linear Regression) Functions
- STATS_BINOMIAL_TEST
- STATS_CROSSTAB
- STATS_F_TEST
- STATS_KS_TEST
- STATS_MODE
- STATS_MW_TEST
- STATS_ONE_WAY_ANOVA
- STATS_T_TEST_*
- STATS_WSR_TEST
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUM
- VAR_POP
- VAR_SAMP
- VARIANCE

Operações de Junção

- As operações de junção retornam uma relação como resultado da combinação de duas outras relações.
- Estas operações adicionais são utilizadas habitualmente em subconsultas na cláusula **from**
- **Condição de junção** – define quais os tuplos que são combinados nas duas relações, assim como quais os atributos que aparecem no resultado da junção.
- **Tipo de junção** – define como tratar os tuplos que não estão relacionados entre si (baseados na condição de junção).

Tipos de Junção
inner join
left outer join
right outer join
full outer join

Condições de Junção
natural
on <predicate>
using (A_1, A_2, ..., A_n)

Operações de Junção

Tipos de Junção
inner join
left outer join
right outer join
full outer join

Condições de Junção
natural
on <predicate>
using (A_1, A_2, ..., A_n)

- É obrigatória a utilização de uma condição de junção nas junções de tipo **outer**. É opcional no **inner join** (na ausência, comporta-se como o produto cartesiano – Oracle não permite inner join sem condição de junção)
- A condição **natural** aparece antes do tipo de junção (por exemplo **natural inner join**). As restantes condições aparecem após o tipo de junção.
- Nas junções com condição **natural**, primeiro aparecem os atributos comuns a ambas as relações, na ordem pela qual aparecem na relação do lado esquerdo. Depois, aparecem os restantes atributos da relação do lado esquerdo, seguidos dos restantes atributos da relação do lado direito.
- Nas junções com condição **using** (A_1, A_2, ..., A_n), primeiro aparecem os atributos A_1, A_2, ..., A_n. Depois, aparecem os restantes atributos da relação do lado esquerdo, seguidos dos restantes atributos da relação do lado direito.

Junção versus Produto Cartesiano

- Listar o nome, número de empréstimo e montante de todos os clientes que fizeram um empréstimo na agência de Perryridge.

  select *borrower.*, amount*
  from *borrower, loan*
  where *borrower.loan_number = loan.loan_number* **and**
  branch_name = 'Perryridge'

  - Versus

  select *customer_name, loan_number, amount*
  from *borrower* **natural inner join** *loan*
  where *branch_name =* 'Perryridge'

- A última separa claramente onde se coloca a origem dos dados de onde se colocam as condições "de filtragem" (seleção)
  - Esta separação não só torna a leitura mais fácil, como pode ser aproveitada para implementações.
- Na última consulta não se pode colocar "*borrower.**" após o **select** pois deixa de ser possível utilizar o nome da tabela nas colunas usadas para fazer a junção natural (isto é, as comuns a ambas as tabelas)

Utilização típica de USING

- Considerem-se as tabelas **cursos(cod_curso,nome)** e **alunos(num_aluno, nome, cod_curso)**

- A pergunta

  select * **from** *cursos* **natural inner join** *alunos*

  é equivalente a

  select *cursos.cod_curso as cod_curso, cursos.nome as nome,*
  alunos.num_aluno
  from *cursos, alunos*
  where cursos.cod_curso = alunos.cod_curso and cursos.nome = alunos.nome

- Enquanto que a pergunta

  select * **from** *cursos* **inner join** *alunos* **using** *(cod_curso)*

  corresponde a

  select *cursos.cod_curso as cod_curso, cursos.nome,*
  alunos.num_aluno, alunos.nome
  from *cursos, alunos*
  where cursos.cod_curso = alunos.cod_curso

- Qual é a que faz sentido?

Relações de Exemplo

- Relação *loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relação *borrower*

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

loan_number	branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Exemplos

customer_name	loan_number
Jones	L-170
Smith	L-230
Hayes	L-155

- **select** * **from** *loan* **inner join** *borrower* **on** *loan.loan_number = borrower.loan_number*

l.loan_number	branch_name	amount	customer_name	b.loan_number
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

select * **from** *loan* **left outer join** *borrower* **on** *loan.loan_number = borrower.loan_number*

l.loan_number	branch_name	amount	customer_name	b.loan_number
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*

Exemplos

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

select * from *loan* natural inner join *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

select * from *loan* natural right outer join *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	null	null	Hayes

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Exemplos

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

select * **from** *loan* **full outer join** *borrower* **using** *(loan_number)*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	null	null	Hayes

Aspectos a ter cuidado

- Ocasionalmente as junções podem ser ambíguas quando estão envolvidas mais do que 2 tabelas:

  select *
  from a **natural left outer join** b **left outer join** c **on** b.c1 = c.c1

- Pode ser interpretado como:

  select *
  from (a **natural left outer join** b) **left outer join** c **on** b.c1 = c.c1

  ou:

  select *
  from a **natural left outer join** (b **left outer join** c **on** b.c1 = c.c1)

- Os operadores associam à esquerda, ou seja, a primeira hipótese é a executada. Na dúvida usar os parêntesis, em particular quando há muitas condições de junção expressas através de ON.
- A condição WHERE aplica-se APÓS as junções terem sido realizadas

Ordem execução JOINS e WHERE

r

A
1
2
4

s

B
1
2
3
6

```
select *
from r left outer join s
on (r.A = s.B AND s.B <= 3)
```

A	B
1	1
2	2
4	

```
select *
from r left outer join s
on (r.A = s.B)
where s.B <= 3
```

A	B
1	1
2	2

Exemplos

Listar todos os clientes que têm uma conta e nenhum empréstimo

select *customer_name*
from *depositor* **left outer join** *borrower* **using(**customer_name**)**
where *loan_number* **is null**

Listar todos os clientes que têm uma conta ou um empréstimo no banco (mas não ambos!)

select *customer_name*
from *depositor* **natural full outer join** *borrower*
where *account_number* **is null or** *loan_number* **is null**

Valores Nulos

- Os tuplos podem conter valores nulos, denotado por *null*, nalguns dos seus atributos.
- *null* significa um valor desconhecido ou que não existe.
- O predicado **is null** pode ser utilizado para testar a existência de valores nulos.
  - E.g. mostrar todos os números de empréstimos com um valor nulo na coluna *amount.*

    select *loan_number*
    from *loan*
    where *amount* **is null**
- O resultado de uma expressão aritmética com *null* é *null*
  - E.g. 5 + null devolve null
- Contudo, as funções de agregação ignoram os nulos
  - A seguir será analisado este assunto mais detalhadamente

Valores Nulos e Lógica Trivalente

- Qualquer comparação com *null* retorna *unknown*
  - *E.g. 5 < null ou null <> null ou null = null*
- Lógica trivalente usando o valor lógico *unknown*:
  - OR: (*unknown* **or** *true*) = *true*, (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: *(true* **and** *unknown) = unknown, (false* **and** *unknown) = false, (unknown* **and** *unknown) = unknown*
  - NOT*: (***not** *unknown) = unknown*
  - "*P* **is unknown"** é verdade se o valor de *P* é *unknown*
- Resultado da condição da cláusula **where** é tratado como *false* quando o seu valor é *unknown*
- A função Oracle NVL(expr,valor_se_nulo) dá o valor de expr quando este não é nulo, e valor_se_nulo caso contrário
- ***Atenção:*** *no Oracle a cadeia vazia comporta-se como null, mas isto não é assim no SQL Standard!*

Valores Nulos e Agregados

- Calcule o total de todos os montantes dos empréstimos

  select sum (*amount)*
  from *loan*

  - A instrução acima ignora montantes nulos
  - Resultado é null se não existir nenhum montante não-nulo
- Todas as funções de agregação exceto **count** ignoram tuplos com valores nulos nos atributos agregados.
- A função de agregação count nunca retorna valores nulos:
  - **count(*)** - Número de linhas do grupo
  - **count**(expr) - Número de linhas com valor da expressão não nula

Exemplos com funções de agregação

A	B
1	1
1	
1	3
2	
3	
3	
3	
4	1
4	2
4	2
4	2
4	3

```
SELECT SUM(a), AVG(DISTINCT a) FROM t
```

SUM(A)	AVG(DISTINCTA)
34	2.5

```
SELECT COUNT(b), SUM(b), AVG(b) FROM t
```

COUNT(B)	SUM(B)	AVG(B)
7	14	2

```
SELECT COUNT(*), COUNT(b), COUNT(DISTINCT b)
FROM t
```

COUNT(*)	COUNT(B)	COUNT(DISTINCT B)
12	7	3

Exemplos com funções de agregação

A	B
1	1
1	
1	3
2	
3	
3	
3	
4	1
4	2
4	2
4	2
4	3

```
SELECT COUNT(DISTINCT b), SUM(DISTINCT b),
AVG(DISTINCT b) FROM t
```

COUNT(DISTINCT B)	SUM(DISTINCT B)	AVG(DISTINCT B)
3	6	2

```
SELECT b, COUNT(*), SUM(b) FROM t GROUP BY b
```

B	COUNT(*)	SUM(B)
1	2	2
	5	
2	3	6
3	2	6

```
SELECT COUNT(*), COUNT(b), SUM(b), AVG(b)
FROM t WHERE a = 3;
```

COUNT(*)	COUNT(B)	SUM(B)	AVG(B)
3	0		

Exemplos com funções de agregação

A	B
1	1
1	
1	3
2	
3	
3	
3	
4	1
4	2
4	2
4	2
4	3

```
SELECT a,b,COUNT(*),COUNT(DISTINCT a), COUNT(b)
FROM t GROUP BY a, b ORDER BY a DESC, b
```

A	B	COUNT(*)	COUNT(DISTINCT A)	COUNT(B)
4	1	1	1	1
4	2	3	1	3
4	3	1	1	1
3		3	1	0
2		1	1	0
1	1	1	1	1
1	3	1	1	1
1		1	1	0

```
SELECT a FROM t GROUP BY a
HAVING MAX(b) IS NULL
```

A
2
3

Cláusula With

- A cláusula **with** permite a definição temporária de relações, cuja definição está disponível na consulta onde a clausula **with** ocorre. Análogo a procedimentos locais de uma linguagem de programação.
- Encontrar as contas de maior saldo

```
with max_balance(value) as
    ( select max (balance)
      from account )
select account_number
from account, max_balance
where account.balance = max_balance.value
```

Exemplo com with

- Quais as agências cuja soma de saldos das suas contas é superior à média da soma dos saldos de todas as agências.

```
with branch_total (branch_name, value) as
   ( select branch_name, sum (balance)
   from account
   group by branch_name),
branch_total_avg(value) as
   ( select avg (value)
   from branch_total )
select branch_name
from branch_total, branch_total_avg
where branch_total.value >= branch_total_avg.value
```

Particularidades do With em Oracle

- O Oracle não permite “column aliasing” na instrução WITH. O comando do acetato anterior deve ser escrito:

```
with branch_total (branch_name, value) as
( select branch_name as branch_name, sum (balance) as value
   from account
   group by branch_name
),
branch_total_avg (value) as
( select avg(value) as value
   from branch_total
)
select branch_name
from branch_total, branch_total_avg
where branch_total.value >= branch_total_avg.value
```