Apontamento SQL - 1ºteste

7 de abril de 2024 19:56

Linguagem de definição de dados (DDL) "CRIAR" BD de acordo com o modelo relacional
Linguagem de manipulação de dados (DML) INTERROGAR e manipular BD de forma declarativa

DATA DEFINITION LANGUAGE (DDL)

Uma tabela SQL é definida recorrendo ao comando **create table**

create table *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$));

- *r* é o nome da relação
- A_i é o nome de um atributo no esquema de relação *r*
- D_i é o tipo de dados dos valores no domínio do atributo A_i

Conversão (implícita) para tipos SQL

CHARACTER(n) CHAR(n)	CHAR(n)
CHARACTER VARYING(n) CHAR VARYING(n)	VARCHAR2(n)
NATIONAL CHARACTER(n) NATIONAL CHAR(n) NCHAR(n)	NCHAR(n)
NATIONAL CHARACTER VARYING(n) NATIONAL CHAR VARYING(n) NCHAR VARYING(n)	NVARCHAR2(n)
INTEGER INT SMALLINT	NUMBER(38)
FLOAT(b) DOUBLE PRECISION REAL	NUMBER

Tipos base:

- **char(n).** Cadeia de caracteres de comprimento fixo *n*.
- **varchar(n).** Cadeia de caracteres de comprimento variável, com o máximo *n* caracteres, especificado pelo utilizador.
- **int.** inteiro (um subconjunto finito dos inteiros, dependente da máquina).
- **smallint.** Inteiro pequeno (um subconjunto do tipo int).
- **numeric(p,d).** Número de vírgula fixa, com precisão de *p* dígitos e com *d* casas decimais.
- **real, double precision.** Números de vírgula flutuante, com precisão dependente da máquina.
- **float(n).** Número de vírgula flutuante, com um mínimo de precisão de *n* dígitos.

Os valores nulos são permitidos em todos os tipos de dados. A declaração de um atributo como **not null** proíbe os valores nulos para esse atributo.

create domain construção em SQL-92 que cria um tipo de dados definido pelo utilizador (não se encontra implementado em Oracle)

create domain *person-name* **char**(20) **not null**

DROP table Remove da BD toda a info sobre a relação (e não apenas os dados)
↳ Caso existam restrições de integridade (ex Chave estrangeira) → cascade constraints
↳ eliminar vistas → drop view

drop table loan cascade constraints

After Table modificar o esquema, ou as restrições sobre relações já existentes
↳ adicionar novos atributos **alter table** *r* **add** *A D*
↳ eliminar atributos de uma relação **alter table** *r* **drop** *A*
↳ add novas restrições

alter table *r* **add constraint** *N R*

em que *N* é um nome dado à nova restrição e R define a restrição. Por exemplo:

alter table *account* **add constraint** *saldo_pos* **check** (*balance* > *0*)

↳ Rem restrições **alter table** *r* **drop constraint** *N*

DATA MANIPULATION LANGUAGE (DML)

SQL é baseada em operações de conjuntos e de álgebra relacional com algumas modificações e extensões

Consulta básica em SQL

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

- A_is representam atributos
- r_is representam relações
- *P* é um predicado.

↳ equivalente em álgebra relacional $\Pi_{A1, A2, ..., An}(\sigma_P (r_1 \times r_2 \times ... \times r_m))$

select → operação projeção da álgebra relacional É utilizada para listar os atributos pretendidos no resultado da consulta

select *branch_name* **from** *loan* $\Pi_{\text{branch-name}}(loan)$ **select** * **from** *loan* → "todos os atributos" (elimina a projeção)

* "_" SQL não permite portanto coloca-se "_" nos nomes *
* SQL não distingue maiúsculas e minúsculas em nomes *

↳ SQL permite duplicados nas relações e nos resultados de consultas
↳ para eliminar os duplicados no resultado, inserir distinct

```
select distinct branch_name
from loan
```

↳ all → indica que os duplicados não devem ser retirados (all é assumido por omissão)

```
select all branch_name
from loan
```

↳ a cláusula pode conter expressões aritméticas (+, -, *, /)

```
select loan_number, branch_name, amount * 100
from loan
```

Where → predicado da seleção da álgebra relacional. Envolve atributos de relações que aparecem na cláusula from

Para encontrar os números de contas da agência da Caparica com saldos superiores a 100.

```
select account_number
from account
where branch_name = 'Caparica' and balance > 100
```

→ "and", "or", e "not"

* Apresentar os números dos empréstimos de montantes entre €90,000 e €100,000 (ou seja, ≥€90,000 e ≤€100,000)

```
select loan_number
from loan
where amount between 90000 and 100000
```

* Para negar a condição pode-se colocar o conetivo **not** antes de **between**

Também existe o operador **in** para testar se um valor pertence a uma lista. E.g.:

```
select *
from branch
where branch_name in ('Lisboa', 'Caparica', 'Almada')
```

* % / → concorda com qualquer subcadeia
_ → concorda com qualquer caracter
|| → concatenação
upper → converssão para maiúsculas
lower → " " minúsculas
*

from , → produto cartesiano

as → Renomeação → nomes c/ espaços tem que usar aspas
↳ o Oracle não gosta mt

if-then-else →

```
select numero,
    case when sexo='M' then 'Senhor ' || nome
         when sexo='F' then 'Senhora '|| nome
    end as titNome
from alunos
```

```
select numero,
    case sexo when 'M' then 'Senhor ' || nome
             when 'F' then 'Senhora '|| nome
    end as titNome
from alunos
```

→ quando a expressão a comparar é a mesma

↳ o else ficaria após todos os when e antes do end

Ordenação de tuplos → order by

```
order by customer_name desc   → desc/asc → omição é asc
order by assets desc nulls last   → null last / null first
```

↳ para ter + que 1 chave de ordenação, separa-se por uma vírgula

Multiconjunto

$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$

```
select A1, A2, ..., An
from r1, r2, ..., rm
where P
```

↳ operações com conjuntos union, intersect, except (minus no Oracle) → para não eliminar os duplicados coloca-se all após as palavras

```
(select customer_name from depositor)
union
(select customer_name from borrower)
```

```
(select customer_name from depositor)
intersect
(select customer_name from borrower)
```

```
(select customer_name from depositor)
except
(select customer_name from borrower)
```

→ minus (Oracle)

Agregação → aplicam-se a multiconjuntos

```
select count (*)
from customer
```

→ conta o nº de linhas

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

→ atributos fora da agregação, têm que aparecer em group by

↳ cláusula Having → predicados aplicados depois, enquanto os do where são antes

```
select branch_name, avg (balance)
from account
group by branch_name
```

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

```
select Fagr1, ... Fagrn
from R1, ..., Rm
where CondW
group by A1, ..., Ag
having CondH
```

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

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

from $R_1, \ldots, R_m$
where $Cond_W$
group by $A_1, \ldots, A_g$
having $Cond_H$

Junções

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

→ como tratar os tuplos que não estão relacionados entre si

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

→ quais os tuplos que são combinados nas duas relações, assim como quais os atributos que aparecem no resultado da junção

antes do tipo ⇒ natural inner join

são os primeiros a aparecer

- 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)
- 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.

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

→ parece um "where"

↳ associam a esquerda, logo são estes os 1 a executar

Nulls

→ is null testa a existência de valores nulos
→ a agregação "ignora" os nulos
→ quando o valor de where é unknown é tratado como false
→ 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

A	B
1	1
1	
1	3
2	
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
2	5	
3	3	6
4	2	6

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

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

```
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

with

→ definição temporária de relações, cuja definição está disponível na consulta onde a cláusula with ocorre. Análogo a procedimentos locais de uma linguagem de programação

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