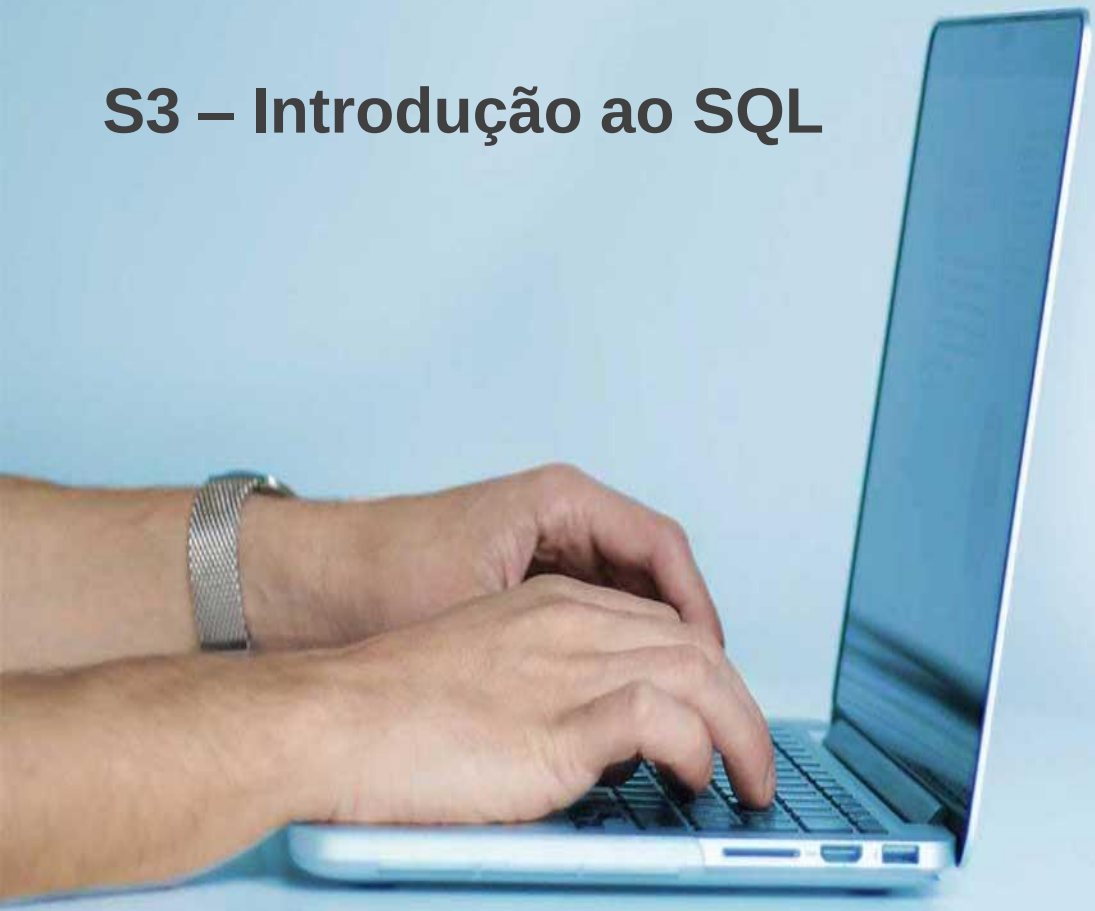
21053 - Fundamentos de Bases de Dados

Professor:

Paulo Pombinho

S3 – Introdução ao SQL



História

- Linguagem IBM Sequel desenvolvida como parte do projeto Sistema R no Laboratório de Investigação IBM de San Jose
- Renomeado Structured Query Language (SQL)
- SQL standard da ANSI e ISO:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (nome da linguagem tornou-se compatível com Y2K!)
  - SQL:2003
- Os sistemas comerciais oferecem a maioria, se não todas,as funcionalidades SQL-92, além de conjuntos de funcionalidades variados de padrões posteriores e características especiais proprietárias.
  - Nem todos os exemplos aqui podem funcionar no seu sistema particular.

Componentes SQL

- DML -- fornece a capacidade de consultar informações a partir da base de dados e de inserir, eliminar e modificar tuplos na base de dados.
- integridade – o DDL inclui comandos para especificar restrições de integridade.
- Definição de Vistas -- O DDL inclui comandos para definir vistas.
- Controlo de transações – inclui comandos para especificar o início e o fim das transações.
- Embedded SQL e SQL Dinâmico -- define como as declarações SQL podem ser incorporadas em linguagens de programação de uso geral.
- Autorização – inclui comandos para especificar direitos de acesso às relações e vistas.

Data Definition Language

A linguagem de definição de dados (DDL) permite a especificação de informação sobre relações, incluindo:

- O esquema para cada relação.
- O tipo de valores associados a cada atributo.
- As restrições de Integridade
- O conjunto de índices a manter para cada relação.
- Informações de segurança e autorização para cada relação.
- A estrutura de armazenamento físico de cada relação no disco.

Tipos de domínio em SQL

- **char(n).** Cadeia de caracteres de comprimento fixo, com comprimento *n* especificado pelo utilizador.
- **varchar(n).** Cadeias de caracteres de comprimento variável, com comprimento máximo*n* especificado pelo utilizador.
- **int.** Inteiro (um subconjunto finito dos inteiros que é dependente da máquina).
- **smallint.** Pequeno inteiro (um subconjunto dependente da máquina do tipo de domínio inteiro).
- **numeric(p,d).** Número de ponto fixo, com precisão especificada pelo utilizador de *p* dígitos, com *d* dígitos à direita do ponto decimal. (ex., numeric(3,1), permite que 44,5 seja armazenada de forma exata, mas não 444,5 ou 0,32)
- **real, double precision.** Ponto flutuante e números de ponto flutuante de dupla precisão, com precisão dependente da máquina.
- **float(n).** Número de ponto flutuante, com precisão especificada pelo utilizador de pelo menos n dígitos.

Criar Tabela

- Uma relação SQL é definida usando o comando**create table** :

  create table *r*

  (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (restrição de integridade$_1$),
  ...,
  (restrição de integridade$_k$))

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

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20)**,**
  dept_name **varchar**(20),
  salary **numeric**(8,2))

Restrições de integridade na criação de tabela

- Tipos de restrições de integridade
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references** *r*
  - **not null**
- SQL impede qualquer atualização à base de dados que viole uma restrição de integridade.
- Exemplo:

create table *instructor* (
ID **char**(5),
name **varchar**(20) **not null,**
dept_name **varchar**(20),
salary **numeric**(8,2),
primary key (*ID*),
foreign key *(dept_name*) **references** *department);*

E mais algumas definições de relação

- **create table** *student* (
  ID **varchar**(5),
  name **varchar**(20) not null,
  dept_name **varchar**(20),
  tot_cred **numeric**(3,0),
  primary key *(ID),*
  foreign key *(dept_name*) **references** *department*);

- **create table** *takes* (
  ID **varchar**(5),
  course_id **varchar**(8),
  sec_id **varchar**(8),
  semester **varchar**(6),
  year **numeric**(4,0),
  grade **varchar**(2),
  primary key *(ID, course_id, sec_id, semester, year)* ,
  foreign key (*ID*) **references** *student,*
  foreign key (*course_id, sec_id, semester, year*) **references** *section*);

E mais ainda

- **create table** *course* (
  course_id **varchar**(8),
  title **varchar(**50),
  dept_name **varchar**(20),
  credits **numeric**(2,0),
  primary key *(course_id),*
  foreign key *(dept_name*) **references** *department*);

Updates às tabelas

- **Inserir**
  - **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **Apagar**
  - Remove todos os tuplos da tabela do aluno
    - **delete from** *student*
- **Apagar tabela**
  - **drop table** *r*
- **Alterar**
  - **alter table** *r* **add** *A D*
    - onde *A* é o nome do atributo a ser adicionado à relação *r* e *D* é o domínio de *A.*
    - A todos os tuplos existentes na relação é atribuído o valor *null* como valor para o novo atributo.
  - **alter table** *r* **drop** *A*
    - onde *A* é o nome de um atributo de relação *r*
    - Apagar atributos não é suportado por muitas bases de dados.

Estrutura de consulta básica

- Uma consulta típica SQL tem a forma:

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

  - A_i representa um atributo
  - R_i representa uma relação
  - P é um predicado.
- O resultado de uma consulta SQL é uma relação.

A cláusula Select

- O **select** lista os atributos desejados no resultado de uma consulta
  - corresponde à operação de projeção da álgebra relacional
  - Exemplo: encontrar os nomes de todos os instrutores:

    select *name*
    from *instructor*
- NOTA: Os nomes SQL são insensíveis à capitalização (ou seja, pode utilizar letras maiúsculas ou minúsculas.)
- Ex., *Name* ≡ *NAME* ≡ *name*
  - Algumas pessoas usam maiúsculas onde estamos a usar Bold/Negrito.

A cláusula Select

- SQL permite duplicações nas relações, bem como nos resultados da consulta.
- Para forçar a eliminação de duplicados, insira a palavra-chave **distinct** após a seleção**.**
- Encontre os nomes do departamento de todos os instrutores e remova duplicados

select distinct *dept_name*
from *instructor*

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

- A palavra-chave **all** especifica que duplicados não devem ser removidos.

select all *dept_name*
from *instructor*

A cláusula Select

- Um asterisco na cláusula selecionada denota "todos os atributos“

  select *
  from *instructor*

- Um atributo pode ser um literal sem a cláusula **from**

  select '437'

  - Resultado é uma tabela com uma coluna e uma única linha com o valor "437"
  - Pode dar à coluna um nome usando:

    select '437' **as** *FOO*

- Um atributo pode ser um literal na cláusula **fro**

  select 'A'
  from *instructor*

  - Resultado é uma tabela com uma coluna e N linhas (número de tuplos na tabela de instrutores), cada linha com o valor "A"

A cláusula Select

- A clausula **select** pode conter expressões aritméticas envolvendo a operação, +, –, ∗, e /, e operando em constantes ou atributos de tuplos.
  - A consulta:

    select *ID, name, salary/12*
    from *instructor*

    devolveria uma relação que é a mesma da relação instrutor, exceto no valor do atributo salário que é dividido por 12.
  - Pode renomear "s*alary/12"* usando a cláusula **as**:

    select *ID, name, salary/12* **as** *monthly_salary*

A cláusula where

- A cláusula **where** especifica as condições que o resultado deve satisfazer
  - Corresponde ao predicado de seleção da álgebra relacional.
- Para encontrar todos os instrutores no departamento de Comp. Sci.

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

- SQL permite a utilização dos conjuntivos lógicos **e, ou,** e **negação**
- Os operadores dos conjuntivos lógicos podem ser expressões envolvendo os operadores de comparação <, <=, >, >=, =, and <>.
- Comparações pode ser aplicado aos resultados de expressões aritméticas
- Para encontrar todos os instrutores no departamento de Comp. Sci. com salário > 70000

```
select name
from instructor
where dept_name = 'Comp. Sci.'  and salary > 70000
```

name
Katz
Brandt

A cláusula from

- A cláusula **from** lista as relações envolvidas na consulta
  - Corresponde ao funcionamento do produto cartesiano da álgebra relacional.
- Encontre o produto Cartesiano *instructor X teaches*

  select ∗
  from *instructor, teaches*

  - gera todos os possíveis pares instructor – teaches, com todos os atributos de ambas as relações.
  - Para atributos comuns (ex. *ID*), os atributos da tabela resultante são renomeados usando o nome de relação (ex., *instructor.ID*)
- Produto cartesiano não muito útil diretamente, mas útil combinado com condição where (operação de seleção em álgebra relacional).

Exemplos

- Encontre os nomes de todos os instrutores que ensinaram algum curso e os course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID*
- Encontrar os nomes de todos os instrutores no departamento de Arte que ensinaram algum curso e o course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID*
    and *instructor. dept_name* = 'Art'

name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Renomear

- O SQL permite renomear relações e atributos usando a cláusula **as**:

  old-name **as** *new-name*

- Encontre os nomes de todos os instrutores que têm um salário mais alto do que algum instrutor em 'Comp. Sci'.
  - **select distinct** *T.name*
    from *instructor* **as** *T, instructor* **as** *S*
    where *T.salary > S.salary* **and** *S.dept_name = 'Comp. Sci.'*

- Cláusula **as** é opcional e pode ser omitida

  instructor **as** *T* ≡ *instructor T*

Exemplo: Self Join

- Relação *emp-super*

person	*supervisor*
Bob	Alice
Mary	Susan
Alice	David
David	Mary

- Encontre o supervisor de "Bob"
- Encontre o supervisor do supervisor de "Bob"
- Pode encontrar todos os supervisores (diretos e indiretos) de "Bob"?

Operações com Strings

- O SQL inclui um operador de correspondência de strings para comparações em cadeias de caracteres. O operador **like** usa padrões que são descritos usando dois caracteres especiais:
  - percentagem (%). O caráter % corresponde a qualquer substring.
  - underscore (_). O caráter _ corresponde a qualquer caráter.
- Encontre os nomes de todos os instrutores cujo nome inclui o substring “dar”.

```
select name
from instructor
where name like '%dar%'
```

- Encontre a string “100%”

```
like '100 \%'  escape  '\'
```

em que acima usamos a barra (\) como o caráter de escape.

Operações com Strings

- Os padrões são sensíveis ao caso.
- Exemplos:
  - 'Intro%' corresponde a qualquer string começando com “Intro”.
  - '%Comp%' corresponde a qualquer corda que contenha “Comp“ como um sub-string.
  - '_ _ _' corresponde a qualquer cadeia de exatamente três caracteres.
  - '_ _ _ %' corresponde a qualquer cadeia de pelo menos três caracteres.
- SQL suporta uma variedade de operações de strings, tais como
  - concatenação (usando “||”)
  - conversão de maiúscula para minúscula (e vice-versa)
  - encontrar comprimento de strings, extrair substrings, etc.

Ordenar a apresentação dos Tuplos

- Lista por ordem alfabética os nomes de todos os instrutores

  select distinct *name*
  from *instructor*
  order by *name*

- Podemos especificar **desc** para a ordem descendente ou **asc** para a ordem ascendente, para cada atributo; ordem ascendente é o padrão.
  - Exemplo: **order by** *name* **desc**
- Pode ordenar em múltiplos atributos
  - Exemplo: **order by** *dept_name, name*

Predicados da cláusuula Where

- SQL inclui um operador de comparação **between**
- Exemplo: Encontre os nomes de todos os instrutores com salário entre $90,000 e $100,000 (I.e., $\geq$ $90,000 e $\leq$ $100,000)
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000
- Comparação de tuplos
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor*.*ID*, *dept_name*) = (*teaches*.*ID*, 'Biology');

Operações de Sets

- Encontre cursos que correram no outono de 2017 ou na primavera de 2018

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2017)
  union
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2018)

- Encontre cursos que correram no outono de 2017 e na primavera de 2018

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2017)
  intersect
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2018)

- Encontre cursos que correram no outono de 2017 mas não na primavera de 2018

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2017)
  except
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2018)

Operaçãos de Sets

- Operações **union**, **intersect**, e **except**
  - Cada uma das operações acima elimina automaticamente duplicados

- Para reter todos os duplicados usar
  - **union all**
  - **intersect all**
  - **except all**.

Valores Null

- É possível que os tuplos tenham um valor nulo, denotado por **null**, para alguns dos seus atributos
- **null** significa um valor desconhecido ou que um valor não existe.
- O resultado de qualquer expressão aritmética envolvendo **null** é **null**
  - Exemplo: 5 + **null** retorna **null**
- O predicado **is null** pode ser usado para verificar se há valores nulos.
  - Exemplo: Encontre todos os instrutores cujo salário é null.

    select *name*
    from *instructor*
    where *salary* **is null**
- O predicado **is not null** devolve true se o valor em que é aplicado não é nulo.

Valores Null

- SQL trata como **unknown** o resultado de qualquer comparação que envolva um valor nulo (que não os predicados **is null** e **is not null**).
  - Exemplo*: 5* < **null** or **null** <> **null** or **null** = **null**
- O predicado numa cláusula **where** pode envolver operações Boolean (**e**, **ou**, **negação**); assim, as definições das operações booleanas precisam ser estendidas para lidar com o valor **unknown**.
  - **and** : *(true* **and** *unknown)* = *unknown,*
    (false **and** *unknown)* = *false,*
    (unknown **and** *unknown)* = *unknown*
  - **or:** (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown)* = *unknown*
- O resultado do predicado da cláusula **where** é tratado como falso se avaliar como *unknown*

Funções Agregadas

- Estas funções operam no multiconjunto de valores de uma coluna de uma relação, e devolvem um valor

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

Exemplos de Funções Agregadas

- Encontre o salário médio dos instrutores no departamento de Informática
  - **select avg** (*salary*)
    from *instructor*
    where *dept_name*= 'Comp. Sci.';
- Encontre o número total de instrutores que lecionam um curso no semestre primavera 2018
  - **select count** (**distinct** *ID*)
    from *teaches*
    where *semester* = 'Spring' **and** *year* = 2018;
- Encontre o número de tuples na relação curso
  - **select count** (*)
    from *course*;

Funções Agregadas – Group By

- Encontre o salário médio dos instrutores em cada departamento
  - **select** *dept_name*, **avg** (*salary*) **as** *avg_salary*
    from *instructor*
    group by *dept_name*;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Funções Agregadas

- Atributos na clausula **select** fora das funções agregadas devem aparecer na lista **group by**
  - /* Consulta errada! */
    select *dept_name*, *ID*, **avg** (*salary*)
    from *instructor*
    group by *dept_name*;

Funções Agregadas – Having

- Encontre os nomes e salários médios de todos os departamentos cujo salário médio seja superior a 42000

  select *dept_name*, **avg** (*salary*) **as** *avg_salary*
  from *instructor*
  group by *dept_name*
  having avg (*salary*) > 42000;

- Nota: predicados na cláusula **having** são aplicados após a formação de grupos, enquanto os predicados na cláusula **where** são aplicados antes de formar grupos

Subconsultas

- SQL fornece um mecanismo para a utilização de subqueries. Uma **subconsulta** é uma expressão **select-from-where** que está colocada dentro de outra consulta.
- A subconsulta pode ser feita na seguinte consulta SQL

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

  como se segue:

  - **From:** r_i pode ser substituído por qualquer subquery válido
  - **Where:** P pode ser substituído por uma expressão da forma:

    B <operação> (subconsulta)

  - **Select:**

    A_i pode ser substituído ser uma subconsulta que gera um único valor.

Operadores de Conjunto

- Find courses offered in Fall 2017 and in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year= 2018);
```

- Find courses offered in Fall 2017 but not in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
        course_id  not in (select course_id
                           from section
                           where semester = 'Spring' and year= 2018);
```

Operadores de Conjunto

- Nomeie todos os instrutores cujo nome não é “Mozart” nem Einstein”

```
select distinct name
from instructor
where  name not in ('Mozart', 'Einstein')
```

- Encontre o número total de alunos (distintos) que tenham frequentado as secções do curso lecionadas pelo instrutor com ID 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
                    (select course_id, sec_id, semester, year
                     from teaches
                     where teaches.ID= 10101);
```

- Nota: A consulta acima pode ser escrita de uma forma muito mais simples. A formulação acima é simplesmente para ilustrar as características do SQL

Comparação de conjuntos – “some”

- Encontre nomes de instrutores com salário superior ao de alguns (pelo menos um) instrutor no departamento de Biologia.

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept name = 'Biology';
```

- Mesma consulta usando clausula > **some**

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept name = 'Biology');
```

Comparação de conjuntos – “all”

- Encontre os nomes de todos os instrutores cujo salário é maior do que o salário de todos os instrutores do departamento de Biologia.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept name = 'Biology');
```

Teste para relações vazias

- O predicado **exists** devolve o valor **true** se a subconsulta no seu argumento é não vazia.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Utilização de “exists”

- Outra forma de especificar a consulta "Encontre todos os cursos lecionados tanto no outono de 2017 como na primavera de 2018“

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
        exists  (select *
        from section as T
        where semester = 'Spring' and year= 2018
                and S.course_id = T.course_id);
```

- **Nome de correlação** – variável S na consulta exterior
- **Subconsulta correlacionado** – a consulta interna

Use of "not exists" Clause

- Encontre todos os alunos que tenham feito todos os cursos oferecidos no departamento de Biologia.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                   except
                    (select T.course_id
                     from takes as T
                     where S.ID = T.ID));
```

  - Primeira subconsulta lista todos os cursos oferecidos em Biologia
  - Segunda subconsulta lista todos os cursos que um aluno específico frequentou

- Note que $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
- Nota: Não é possível escrever esta consulta usando = all e as suas variantes

Teste de ausência de tuplos duplicados

- O predicado **unique** testes se uma subconsulta tem qualquer tuplo duplicado no seu resultado.
- O predicado **unique** retorna “true" se uma dada subconsulta não contém duplicados.
- Encontre todos os cursos que foram oferecidos no máximo uma vez em 2017

```
select T.course_id
 from course as T
 where unique ( select R.course_id
                from section as R
                where T.course_id= R.course_id
                      and R.year = 2017);
```

Subconsultas na clausula From

- SQL permite que uma subconsulta seja usada na cláusula **from**
- Encontrar os salários médios dos instrutores dos departamentos onde o salário médio é maior do que $42,000."

```
select dept_name, avg_salary
from ( select dept_name, avg (salary) as avg_salary
        from instructor
        group by dept_name)
where avg_salary > 42000;
```

- Note que não precisamos de usar **having**
- Outra forma de escrever a consulta acima

```
select dept_name, avg_salary
from ( select dept_name, avg (salary)
        from instructor
        group by dept_name)
        as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

Cláusula With

- A cláusula **with** fornece uma forma de definir uma relação temporária cuja definição está disponível apenas para a consulta em que **with** ocorre.
- Encontre todos os departamentos com o orçamento máximo

```
with max_budget (value) as
        (select max(budget)
         from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

Consultas Complexas utilizando With

- Encontre todos os departamentos onde o salário total é superior à média do salário total em todos os departamentos
-

```
with dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

Subconsulta escalar

- Subconsulta escalar é uma consulta em que um único valor é esperado
- Listar todos os departamentos juntamente com o número de instrutores em cada departamento

```
select dept_name,
        ( select count(*)
          from instructor
          where department.dept_name = instructor.dept_name)
        as num_instructors
from department;
```

- Erro de tempo de execução se subconsulta devolver mais de um tuplo

Modificação da Base de Dados

- Remoção de tuplos de uma dada relação.
- Inserção de novos tuplos numa dada relação
- Atualização de valores em alguns tuplos numa determinada relação

Remoção

- Apagar todos os instrutores

  delete from *instructor*

- Apagar todos os instrutores do Departamento de Finanças

  delete from *instructor*
  where *dept_name*= 'Finance';

- *Elimine todos os tuplos na relação instrutor para os instrutores associados a um departamento localizado no edifício Watson.*

  delete from *instructor*
  where *dept name* **in** (**select** *dept name*
  from *department*
  where *building* = 'Watson');

Remoção

- Eliminar todos os instrutores cujo salário seja inferior ao salário médio dos instrutores
- **delete from** *instructor*
  where *salary* < (**select avg** (*salary*)
  from *instructor*);

  - Problema: à medida que eliminamos tuples de *instructor*, o salário médio muda!
  - Solução utilizada em SQL:
    1. Primeiro, calcular **avg** (salary) e encontrar todos os tuplos a apagar
    2. Em seguida, apague todos os tuplos encontrados acima (sem recalcular **avg** ou retestar o tuplos)

Inserção

- Adicione um novo tuplo ao curso

  insert into *course*
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- ou de forma equivalente

  insert into *course* (*course_id*, *title*, *dept_name*, *credits*)
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- Adicione um novo tuplo em *student* com *tot_creds* definido para null

  insert into *student*
  values ('3003', 'Green', 'Finance', *null*);

Inserção

- Faça de cada aluno do departamento de Música que ganhou mais de 144 créditos um instrutor no departamento de Música com um salário de $18.000.

```
insert into instructor
    select ID, name, dept_name, 18000
    from   student
    where  dept_name = 'Music' and total_cred > 144;
```

- A declaração **select from where** é avaliados totalmente antes de qualquer um dos seus resultados ser inserido na relação.

Caso contrário, consultas como

```
insert into table1 select * from table1
```

iria causar problemas

Updates

- Dê um aumento salarial de 5% a todos os instrutores

```
update instructor
   set salary = salary * 1.05
```

- Dê um aumento salarial de 5% aos instrutores que ganham menos de 70000

```
update instructor
   set salary = salary * 1.05
  where salary < 70000;
```

- Dê um aumento salarial de 5% aos instrutores cujo salário seja inferior à média

```
update instructor
 set salary = salary * 1.05
 where salary <  (select avg (salary)
                  from instructor);
```

Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - The order is important
  - Can be done better using the **case** statement (next slide)

Cláusula case para Updates condicionais

- Mesma consulta de antes, mas com uso do **case**

```
update instructor
    set salary = case
                    when salary <= 100000 then salary * 1.05
                    else salary * 1.03
                 end
```

Updates com subconsultas escalares

- Recalcular e atualizar valor tot_creds para todos os alunos

```
update student S
set tot_cred = (select sum(credits)
                from takes, course
                where takes.course_id = course.course_id  and
                      S.ID= takes.ID.and
                      takes.grade <> 'F' and
                      takes.grade is not null);
```

- Colocar *tot_creds* em null para os alunos que não frequentaram ainda qualquer curso
- Em vez de **sum**(*credits*), usar:

```
case
    when sum(credits) is not null then sum(credits)
    else 0
end
```

21053 - Fundamentos de Bases de Dados

UNIVERSIDADE ABERTA www.uab.pt

Professor:

Paulo Pombinho

Bom Estudo!