Disciplina de Modelação de Dados em Engenharia

Material de Apoio às Aulas Teóricas (Português/Ingles)

Resp. Disciplina: **Luis Camarinha Matos**
João Rosas
Yves Rybarczyk
Pedro Santana

DEE / FCT / UNL
Secção de Robótica e Manufactura Integrada

Tópicos para hoje

- Regras de integridade; especificando restrições
- Normalização das BD
- SQL (2ª parte)
- PL/SQL

Especificando restrições (1) “constraint”

- Cláusula ‘Constraint’: faz parte dos comandos ‘create table’ ou ‘alter table’.
- Formato:
- CONSTRAINT ‘nome_constr’ espefic_constr

- As especificações de constrangimento são as seguintes:
  - **PRIMARY KEY** - O Atributo é único, com restrição ‘not null’ e indexado.
  - **UNIQUE** - O Atributo é único e indexado mas não é chave.
  - **NOT NULL** - O valor do atributo não pode estar em branco.
  - **REFERENCES** - Constrangimento de chave externa. O valor do atributo precisa de corresponder com o valor do atributo da tabela que referência.’

Especificando restrições (2) exemplo

```
Create table aluno (
    numero integer primary key, -- not null, unique, indexed by default
    Nome varchar2(40) not null,
    Sexo   char(1) check (sexo='M' or sexo='F'),
    constraint numero_valido check (numero> 0 and numero<99999),
    constraint nome_unico unique(nome) -- unique and indexed by default
    but not a key
)
```

```
CREATE INDEX nome_aluno_idx
  ON alunos(nome);
```

Propriedades das relações
Regra 1

- Cada coluna/atributo deve possuir um nome que deverá ser único.
  - A ordem das linhas ou atributos não têm qualquer significado, nem necessitam de ordenação.
  - Pois sendo um conjunto de atributos, os conjuntos não necessitam de ordenação.
  - **Cada ai tem que ser diferente dos restantes e, …**
  - **tem-se que {a1,a2,a3,a4,a5} = {a3,a5,a1,a2,a4}**

Tabela

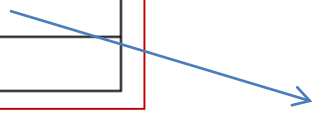
a1	a2	a3	a4	a5
1	a	x	100	Frio
2	b	y	200	Morno
3	c	z	300	quente

Propriedades das relações
Regra 2

- A ordem das linhas (tuplos) na tabela não deve ter qualquer significado.
  - Caso contrário se essa ordem fosse alterada, a informação perderia o seu significado (ver exemplo abaixo)
  - Neste exemplo, a solução passa pela adição de um novo atributo que especifica a ordem da operação para fabricar o artigo.

Plan. Processos

	ArtigoID	Tarefa	Sequência
...	...	...	...
#1011	A1	Cortar material	1
#1012	A1	tornear	2
#1013	A1	frezar	3
#1014	A1	acabamento	4
#1015	A1	esmerilar	3.5
...	...	...	...



FAZER um TRIGGER para reordenar as tarefas

Propriedades das relações
Regra 3

- Nenhuma linha (tuplo) deve ser exactamente igual a outra linha da tabela.
  - Caso acontecesse, a tabela deixaria de ter as propriedades de um conjunto, pois um conjunto não contêm elementos repetidos.
  - Já não seria tão fácil utilizar álgebra para manipular informação (união, intercepção, diferença, projecção) -> (dado mais à frente a "Álgebra relacional").
  - Tal facto levaria a uma maior complexidade e ineficiência dos SGBD e correspondentes aplicações clientes.
  - Basicamente, não pode haver informação repetida numa tabela.

Propriedades das relações
Regra 4

- Só pode existir apenas um valor em cada intercepção linha/coluna de uma tabela.
  - Cada campo de um registo tem que possuir apenas um valor único, ou seja, um valor atómico.

Tabela

a1	a2	a3	a4	a5
(1,'aa')	a	x	100	Frio
(2,'bb')	b	y	200	Morno
(3,'cc')	c	z	300	quente

Integridade dum modelo relacional

- Garantir que o modelo que representa os dados seja uma expressão exacta da realidade.
- A base de dados deverá, ao longo do tempo, garantir a sua exactidão, de forma manter a sua utilidade.
- Para que a integridade se mantenha, é necessário aplicar as regras seguintes -> prox. slide

Regras de integridade

- Regra de Integridade 1 (integridade das entidades):
  - As chaves primárias não podem ser nulas, ou alguma das suas componentes ser nula (para o caso de chaves primárias compostas a partir de chaves externas).

- Regra de Integridade 2 (integridade referencial)
  - Uma chave externa de uma relação, deve existir também como chave primária de outra relação, caso contrário essa chave externa deverá possuir um valor nulo.

Normalização

- O processo conhecido como normalização, assegura que as tabelas, quando agrupadas de uma determinada forma, constituindo um sistema de informação, assegura que:
  - Evitar dados redundantes (estes devem ser armazenados uma única vez e numa única localização)
  - Assegurar que a informação se mantem consistente.

- Constitui um método formal usado na identificação das relações baseadas nas suas chaves primárias e nas dependências funcionais entre os atributos.

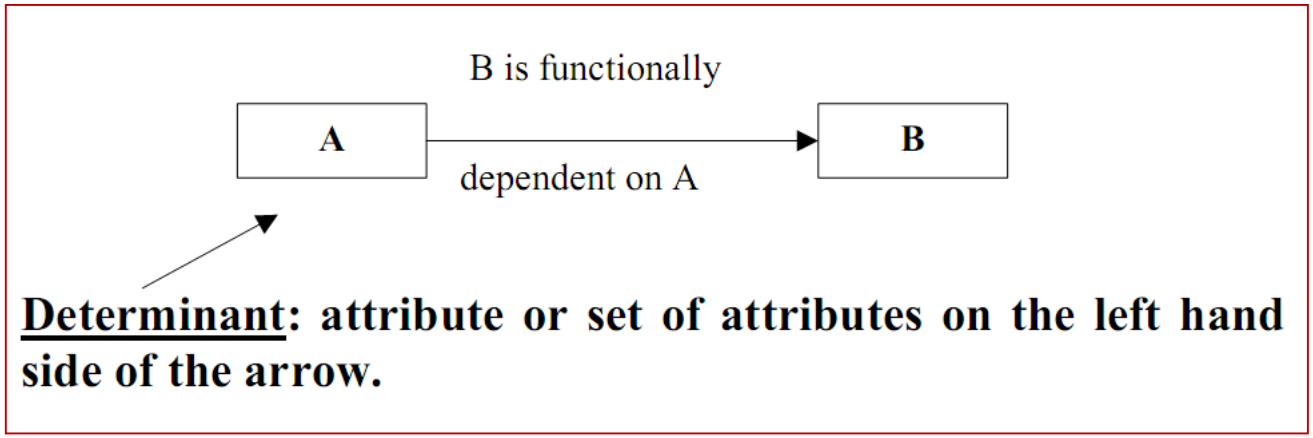
- Dependência funcional: Descreve o relacionamento entre os atributos duma relação/tabela.

- Sejam A e B são atributos de uma relação R. B depende de A (A->B), se cada valor de A em R existir exactamente um valor de B em R.

Determinante

- Identificar a chave candidata duma relação: é o atributo (ou grupo de attributos) que identificam individualmente cada linha duma relação.

- Todos os atributos que não pertencem à chave primária deverão ser dependentes da chave primária referida



Processo de normalização

- Não normalizada (UNF): tabela que contem grupos repetitivos de informação.

- Grupos repetitivos: um ou vários atributos duma tabela que contém multiplos valores para a mesma chave primária.

1NF

- First normal form (1NF): Uma relação em que a intercepção duma linha com uma coluna contem apenas um valor.

aluno	Nome	Nasc	Id_dis	Discipl	Ano	Nota
2020	Ramalho	1940	1	Inglês	3	18
2020	Ramalho	1940	2	Gestão	4	16
2030	Ramalho	1940	3	Estatística	2	10
2030	Soares	1935	4	Francês	1	16
2030	Soares	1935	2	Gestão	4	15
2040	Sampaio	1942	1	Inglês	3	14

UNF → 1NF

- Colocar os valores apropriados nas colunas vazias.
- Colocar os dados repetidos, com a chave primária, numa relação/tabela à parte.
- Identificar a chave primária para cada uma das novas relações

=> alunos: (aluno, nome, nascimento)

⇒disciplina: (id_dis, disciplina)

=> frequência: (aluno, id_dis, ano, nota)

2NF

- Uma relação na 1NF está também na 2NF, se todos os atributos que não são chave primária dependerem da totalidade da chave primária.

- Aplica-se a relações com chaves compostas. Uma relação só com um atributo na chave primária está pelo menos na 2NF.

Ex: fornecedor(id_forn, nome, artigo, morada, preco)

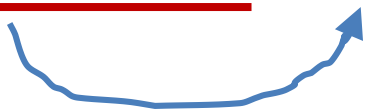
1NF → 2NF:

- Remover dependencias parciais: os atributos dependentes são removidos da relação e colocados numa relação à parte, incluíndo uma cópia do atributo determinante.

Ex: fornecedor(id_forn, nome, artigo, morada, preco)

⇒fornecedor(id_forn, nome, morada)

⇒ fornecimento(id_forn, artigo, preco)

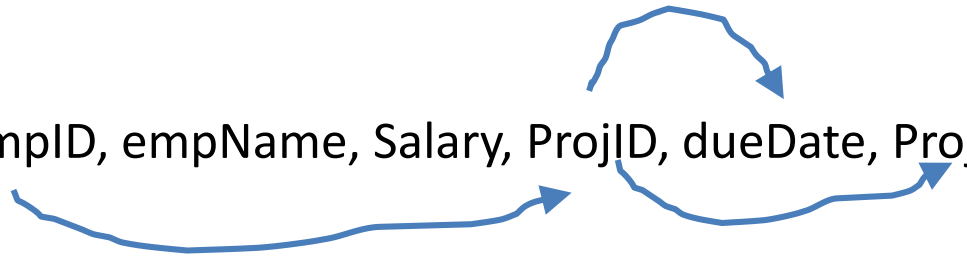


Third normal form (3NF):

- Tem que ser uma relação que já é 1NF e 2NF e em que…

- todos os atributos que não estejam dependentes da chave primária devem ser eliminados e colocados numa tabela à parte.

- Ex: emp: (empID, empName, Salary, ProjID, dueDate, ProjName)



2NF → 3NF

- Remover todos os atributos que não dependam directamenta da chave primária, colocando-os numa relação separada.

Ex: emp: (empID, empName, Salary, ProjID, dueDate, ProjName)

=>emp: (empID, empName, Salary)

⇒Proj: (proID, ProjName, dueDate)

⇒Assignment: (empID, projID)

Resultado da normalização

- Um processo de normalização decompõem a relação/tabela original em várias relações (através de uma série projecções – ver algebra relacional).

- Esta decomposisão, feita sem perdas de informação, é reversivel mediante o operador join (select/join).

Introdução ao PL/SQL

- SQL – Structured Query Language (SQL) is the language used to manipulate relational databases. SQL is tied very closely with the relational model.

  http://cisnet.baruch.cuny.edu/holowczak/oracle/sqlplus/

- PL/SQL: is a procedural language extension to SQL. Its purpose is to combine database language and procedural programming language. The basic unit in PL/SQL is called a block, which is made up of three parts: a declarative part, an executable part, and an exception-building part.

SQL Statements

- The following is an alphabetical list of SQL statements that can be issued against an Oracle database. These commands are available to any user of the Oracle database. These are the most commonly used:
- **ALTER** - Change an existing table, view or index definition
- **AUDIT** - Track the changes made to a table
- **COMMENT** - Add a comment to a table or column in a table
- **COMMIT** - Make all recent changes permanent
- ***CREATE*** - Create new database objects such as tables or views
- ***DELETE*** - Delete rows from a database table
- ***DROP*** - Drop a database object such as a table, view or index
- **GRANT** - Allow another user to access database objects such as tables or views
- ***INSERT*** - Insert new data into a database table
- **No AUDIT** - Turn off the auditing function
- **REVOKE** - Disallow a user access to database objects such as tables and views
- **ROLLBACK** - Undo any recent changes to the database
- ***SELECT*** - Retrieve data from a database table
- ***UPDATE*** - Change the values of some data items in a database table

http://cisnet.baruch.cuny.edu/holowczak/oracle/sqlplus/

Create statement: books

```
CREATE TABLE books (
    book_id                 VARCHAR2(20),
    title                   VARCHAR2(50),
    author_last_name VARCHAR2(30),
    author_first_name       VARCHAR2(30),
    rating NUMBER,
    CONSTRAINT last_name_not_null       NOT NULL,
    CONSTRAINT title_not_null           NOT NULL,
    CONSTRAINT books_pk         PRIMARY KEY (book_id),
    CONSTRAINT rating_1_to_10 CHECK (rating IS NULL OR
                                    (rating >= 1 and rating <= 10)),
    CONSTRAINT author_title_unique UNIQUE (author_last_name, title));
```

Column Name	Type	Size	Other Information and Notes
book_id	VARCHAR2	20	Primary Key (Automatically checks Not Null; an index is also created on the primary key column. This is the Dewey code or other book identifier.)
title	VARCHAR2	50	Not Null
author_last_name	VARCHAR2	30	Not Null
author_first_name	VARCHAR2	30	
rating	NUMBER		(Librarian's personal rating of the book, from 1 (poor) to 10 (great))

Create statement: patrons

```
CREATE TABLE patrons (
  patron_id NUMBER,
  last_name VARCHAR2(30)
    CONSTRAINT patron_last_not_null NOT NULL,
  first_name VARCHAR2(30),
  street_address VARCHAR2(50),
  city_state_zip VARCHAR2(50),
  location MDSYS.SDO_GEOMETRY,
  CONSTRAINT patrons_pk PRIMARY KEY (patron_id));
```

Column Name	Type	Size	Other Information and Notes
patron_id	NUMBER		Primary Key. (Unique patron ID number, with values to be created using a sequence that you will create)
last_name	VARCHAR2	30	Not Null
first_name	VARCHAR2	30	
street_address	VARCHAR2	30	
city_state_zip	VARCHAR2	30	

Column Name	Type	Other Information and Notes
location	Complex type Schema: MDSYS Type: SDO_GEOMETRY	(Oracle Spatial geometry object representing the patron's geocoded address)

Create statement: transactions

```
CREATE TABLE transactions (
  transaction_id NUMBER,
  patron_id CONSTRAINT for_key_patron_id  REFERENCES patrons(patron_id),
  book_id    CONSTRAINT for_key_book_id      REFERENCES books(book_id),
  transaction_date DATE          CONSTRAINT tran_date_not_null NOT NULL,
  transaction_type NUMBER    CONSTRAINT tran_type_not_null NOT NULL,
  CONSTRAINT transactions_pk PRIMARY KEY (transaction_id));
```

Column Name	Type	Size	Other Information and Notes
transaction_id	NUMBER		Primary Key. (Unique transaction ID number, with values to be created using a trigger and sequence that will be created automatically)
patron_id	NUMBER		(Foreign key; must match a patron_id value in the PATRONS table)
book_id	VARCHAR2	20	(Foreign key; must match a book_id value in the BOOKS table)
transaction_date	DATE		(Date and time of the transaction)
transaction_type	NUMBER		(Numeric code indicating the type of transaction, such as 1 for checking out a book)

Create a sequence

(to generate unique numeric values)

```
CREATE SEQUENCE patron_id_seq
   START WITH 100
   INCREMENT BY 1;


INSERT INTO patrons VALUES
    (patron_id_seq.nextval,
   'Smith', 'Jane', '123 Main Street', 'Mytown, MA
    01234', null);
```

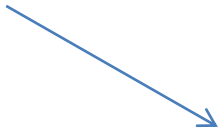
Insert data into the tables

```
INSERT INTO books VALUES ('A1111', 'Moby Dick', 'Melville', 'Herman', 10);
INSERT INTO books VALUES ('A2222', 'Get Rich Really Fast', 'Scammer',
        'Ima', 1);
INSERT INTO books VALUES ('A3333', 'Finding Inner Peace', 'Blissford',
        'Serenity', null);
INSERT INTO books VALUES ('A4444', 'Great Mystery Stories', 'Whodunit',
        'Rodney', 5);
INSERT INTO books VALUES ('A5555', 'Software Wizardry', 'Abugov', 'D.',
        10);

INSERT INTO patrons VALUES  (patron_id_seq.nextval,
  'Smith', 'Jane', '123 Main Street', 'Mytown, MA 01234', null);
INSERT INTO patrons VALUES  (patron_id_seq.nextval,
  'Chen', 'William', '16 S. Maple Road', 'Mytown, MA 01234', null);
INSERT INTO patrons VALUES  (patron_id_seq.nextval,
  'Fernandez', 'Maria', '502 Harrison Blvd.', 'Sometown, NH 03078', null);
INSERT INTO patrons VALUES  (patron_id_seq.nextval,
  'Murphy', 'Sam', '57 Main Street', 'Mytown, MA 01234', null);

INSERT INTO transactions (patron_id, book_id,
 transaction_date, transaction_type)
 VALUES (100, 'A1111', SYSDATE, 1);
INSERT INTO transactions (patron_id, book_id,
 transaction_date, transaction_type)
 VALUES (100, 'A2222', SYSDATE, 2);
INSERT INTO transactions (patron_id, book_id,
 transaction_date, transaction_type)
 VALUES (101, 'A3333', SYSDATE, 3);
```

```
INSERT INTO transactions (patron_id, book_id,
  transaction_date, transaction_type)
  VALUES (101, 'A2222', SYSDATE, 1);
INSERT INTO transactions (patron_id, book_id,
  transaction_date, transaction_type)
  VALUES (102, 'A3333', SYSDATE, 1);
INSERT INTO transactions (patron_id, book_id,
  transaction_date, transaction_type)
  VALUES (103, 'A4444', SYSDATE, 2);
INSERT INTO transactions (patron_id, book_id,
  transaction_date, transaction_type)
  VALUES (100, 'A4444', SYSDATE, 1);
INSERT INTO transactions (patron_id, book_id,
  transaction_date, transaction_type)
  VALUES (102, 'A2222', SYSDATE, 2);
INSERT INTO transactions (patron_id, book_id,
  transaction_date, transaction_type)
  VALUES (102, 'A5555', SYSDATE, 1);
INSERT INTO transactions (patron_id, book_id,
  transaction_date, transaction_type)
  VALUES (101, 'A2222', SYSDATE, 1);
```



Sugestão: criar uma sequence para transaction_id

Create view (1)

```
Create view patrons_trans_view as
SELECT p.patron_id,
      p.last_name,
      p.first_name,
      t.transaction_type,
      t.transaction_date
  FROM patrons p, transactions t
 WHERE p.patron_id = t.patron_id
 ORDER BY p.patron_id, t.transaction_type;

Select * from patrons_trans_view;
```

Create view (2)

```
create view empregado as
select empno as numero, ename as nome, job as categoria,
    sal as salario
from scott.emp;

select numero, nome, salario from empregado
WHERE rownum between 1 and 4;
```

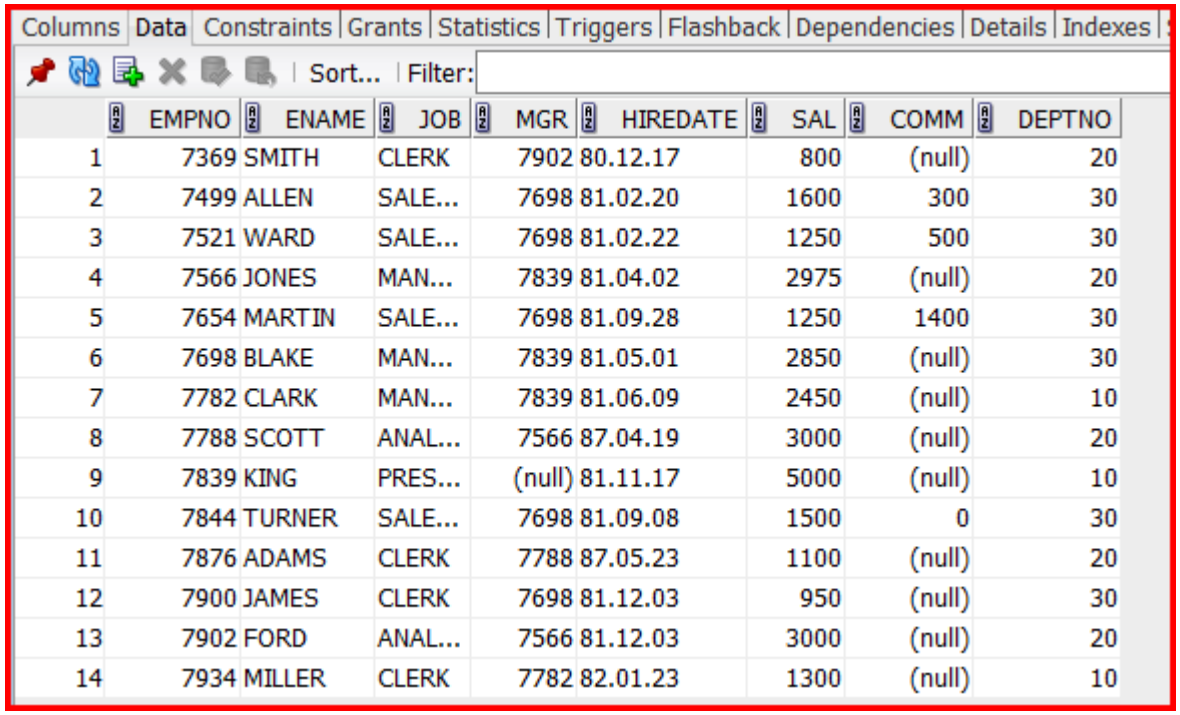
NUMERO	NOME	SALARIO
7369	SMITH	800
7499	ALLEN	1600
7521	WARD	1250
7566	JONES	2975

Select

```
SELECT p.patron_id,
       p.last_name,
       p.first_name,
       t.transaction_type,
       t.transaction_date
  FROM patrons p, transactions t
 WHERE p.patron_id = t.patron_id
 ORDER BY p.patron_id, t.transaction_type;
```

Tabela empregados

- select * from scott.emp;



Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Indexes

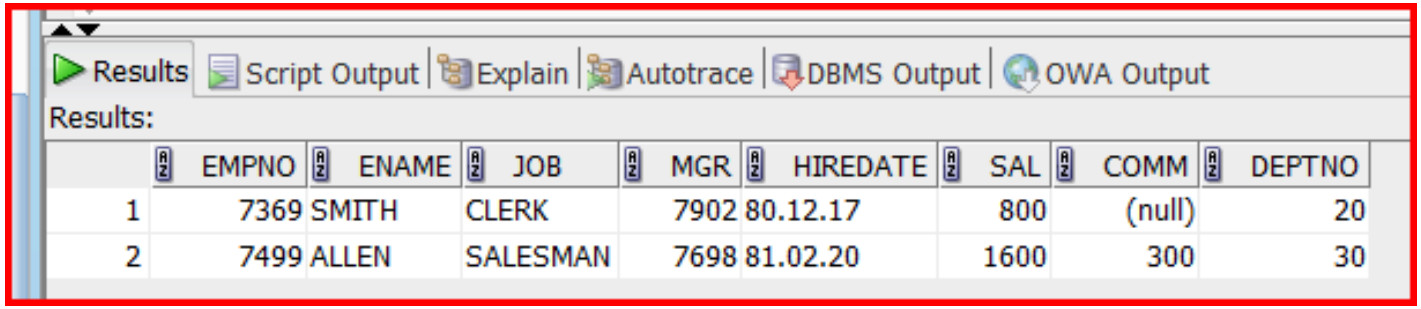
Sort... | Filter:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	80.12.17	800	(null)	20
2	7499	ALLEN	SALE...	7698	81.02.20	1600	300	30
3	7521	WARD	SALE...	7698	81.02.22	1250	500	30
4	7566	JONES	MAN...	7839	81.04.02	2975	(null)	20
5	7654	MARTIN	SALE...	7698	81.09.28	1250	1400	30
6	7698	BLAKE	MAN...	7839	81.05.01	2850	(null)	30
7	7782	CLARK	MAN...	7839	81.06.09	2450	(null)	10
8	7788	SCOTT	ANAL...	7566	87.04.19	3000	(null)	20
9	7839	KING	PRES...	(null)	81.11.17	5000	(null)	10
10	7844	TURNER	SALE...	7698	81.09.08	1500	0	30
11	7876	ADAMS	CLERK	7788	87.05.23	1100	(null)	20
12	7900	JAMES	CLERK	7698	81.12.03	950	(null)	30
13	7902	FORD	ANAL...	7566	81.12.03	3000	(null)	20
14	7934	MILLER	CLERK	7782	82.01.23	1300	(null)	10

Select (2)

- select * from scott.emp

where empno>=7300 and empno<=7500;



	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	80.12.17	800	(null)	20
2	7499	ALLEN	SALESMAN	7698	81.02.20	1600	300	30

Media dos salários

- select avg(sal) from scott.emp;

```
AVG(SAL)
----------------------
2073,214285714285714285714285714285714286
1 rows selected
```

- select avg(sal) as media from scott.emp;

```
media
----------------------
2073,214285714285714285714285714285714286
1 rows selected
```

Empregados com salários acima da média (sortudos!)

- select avg(sal) from scott.emp;

(2073,214285714285714285714285714285714286) <-slide anterior

- SELECT * from scott.emp

where sal> (select avg(sal) from scott.emp);

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7566	JONES	MANAGER	7839	81.04.02	2975	(null)	20
2	7698	BLAKE	MANAGER	7839	81.05.01	2850	(null)	30
3	7782	CLARK	MANAGER	7839	81.06.09	2450	(null)	10
4	7788	SCOTT	ANALYST	7566	87.04.19	3000	(null)	20
5	7839	KING	PRESIDENT	(null)	81.11.17	5000	(null)	10
6	7902	FORD	ANALYST	7566	81.12.03	3000	(null)	20

Numero de empregados com salário abaixo da méda

- Select count(*) from scott.emp;

```
COUNT(*)
---------------------
14
```

- Select count(*) from scott.emp

Where sal< (select avg(sal) from scott.emp);

```
COUNT(*)
---------------------
8
```

Empregado com salário mais baixo/alto

- Select max(sal) from scott.emp;

```
MAX(SAL)
----------
5000
```

- Select min(*) from scott.emp;

```
MIN(SAL)
----------
800
```

```
select empno, ename, sal as salary
from scott.emp
where sal = (
  select max(sal) from scott.emp
);
```

```
EMPNO                ENAME      SALARY
-------------------- ---------- --------------------
7839                 KING       5000

1 rows selected
```

Procedures

- Aumenta salário dum empregado

```
CREATE OR REPLACE PROCEDURE aumenta_sal (p_empno IN emp.empno%TYPE) IS
BEGIN
   UPDATE scott.emp
   SET sal = sal * 1.10
   WHERE empno = p_empno;
END aumenta_sal;
```

P_empno est du même type que EMPNO défini dans la table emp

Passe le numero de l'employé en paramêtre

```
SELECT empno, sal
  FROM scott.emp;

EMPNO      SAL
---------- ----------
7839       5000
7698       2850
7782       2450
```

```
CALL AUMENTA_SAL(7839);
```

Appel de la procedure

```
SELECT empno, sal
  FROM scott.emp;

    EMPNO      SAL
---------- ----------
     7839       5500
     7698       2850
     7782       2450
```

http://www.linhadecodigo.com.br/artigo/335/PL_SQLProcedures-e-Fun%C3%A7%C3%B5es.aspx

If-then-else

Fonction qui renvoie la tranche de revenu (‘income’) à partir du nom de l’employé

```
CREATE OR REPLACE Function IncomeLeve (name_in IN varchar2)
        RETURN varchar2
    IS
        monthly_value number(6);
        ILevel varchar2(20);
   cursor c1 is
          select monthly_income
          from employees
          where name = name_in;
BEGIN
    open c1;
      fetch c1 into monthly_value;
    close c1;
    IF monthly_value <= 4000 THEN
      ILevel := 'Low Income';
    ELSIF monthly_value > 4000 and monthly_value <= 7000 THEN
      ILevel := 'Avg Income';
    ELSIF monthly_value > 7000 and monthly_value <= 15000 THEN
      ILevel := 'Moderate Income';
    ELSE
      ILevel := 'High Income';
    END IF;
    RETURN ILevel;
END;
```

La fonction retourne une string

Déclaration de variables locales

Curseur qui parcourt chaque ligne de la table ‘employees’ afin de lire la valeur contenue dans l’attribut ‘monthly_income’ de cette table

Extraction de monthly_income dans monthly_value

select IncomeLeve(‘King’) from employees;

↳ Avg Income

c1 ⇨

: :

employees

Name	Monthly_income
Smith	800
Allen	1600
King	5000
…	…

http://www.techonthenet.com/oracle/loops/if_then.php

Case statement

- Estes blocos são equivalentes

```
select table_name,
CASE owner
  WHEN 'SYS' THEN 'The owner is SYS'
  WHEN 'SYSTEM' THEN 'The owner is SYSTEM'
  ELSE 'The owner is another value'
END
from all_tables;
```

```
select table_name,
CASE
  WHEN owner='SYS' THEN 'The owner is SYS'
  WHEN owner='SYSTEM' THEN 'The owner is
SYSTEM'
  ELSE 'The owner is another value'
END
from all_tables;
```

```
IF owner = 'SYS' THEN
     result := 'The owner is SYS';
ELSIF owner = 'SYSTEM' THEN
    result := 'The owner is SYSTEM'';
ELSE
    result := 'The owner is another value';
END IF;
```

http://www.techonthenet.com/oracle/

Loop statments

- Sintaxe para os loops:

```
LOOP
    {.statements.}
END LOOP;
```

```
DBMS_OUTPUT.ENABLE;
LOOP
    monthly_value := daily_value * 31;
    DBMS_OUTPUT.PUT_LINE('valor do mes='  || monthly_value  );
    EXIT WHEN monthly_value > 4000;
END LOOP;
DBMS_OUTPUT.DISABLE;
```

http://www.techonthenet.com/oracle/

For statments

- Sintaxe para os for statements

```
FOR loop_counter IN [REVERSE] lowest_number..highest_number
LOOP
    {.statements.}
END LOOP;
```

```
FOR Lcntr IN 1..20
LOOP
    LCalc := Lcntr * 31;
END LOOP;
```

```
FOR Lcntr IN REVERSE 1..15
LOOP
    LCalc := Lcntr * 31;
END LOOP;
```

http://www.techonthenet.com/oracle/

CURSOR FOR Loop

- Sintaxe:

```
FOR record_index in cursor_name
LOOP
    {.statements.}
END LOOP;
```

select totalIncome('CLERK') from scott.emp;

```
CREATE OR REPLACE Function TotalIncome
   (job_in IN   varchar2)
   RETURN varchar2
IS
   total_val number(6);

   cursor c1 is
     select sal
     from scott.emp
     where job = job_in;

BEGIN
     total_val := 0;
     FOR toto in c1
     LOOP
         total_val := total_val + toto.sal;
     END LOOP;

     RETURN total_val;
END;
```

Incremente cette valeur uniquement quand l'employé a un job = 'CLERK'

Fonction qui renvoie la somme des salaires pour un job determiné

=

select sum(sal) from scott.emp where job = 'CLERK'

http://www.techonthenet.com/oracle/loops/cursor_for.php

While Loop

- Sintaxe:

```
WHILE condition
LOOP
    {.statements.}
END LOOP;
```

```
WHILE monthly_value <= 4000
LOOP
    monthly_value := daily_value * 31;
END LOOP;
```

```
LOOP
    monthly_value := daily_value * 31;
    EXIT WHEN monthly_value > 4000;
END LOOP;
```

http://www.techonthenet.com/oracle/

Sequencias

- Criar um “autonumber”
- Útil para gerar chaves primárias

```
CREATE SEQUENCE
sequence_name
   MINVALUE value
   MAXVALUE value
   START WITH value
   INCREMENT BY value
   CACHE value;
```

```
CREATE SEQUENCE supplier_seq
   MINVALUE 1
   START WITH 1
   INCREMENT BY 1
   CACHE 20;
```

```
INSERT INTO suppliers(supplier_id, supplier_name)
VALUES (supplier_seq.nextval, 'Kraft Foods');
```

```
Select max(empNO)+1 as nextVal from scott.emp;
```

Uso de “Cursors”

- Sintaxe

```
CURSOR cursor_name
IS
   SELECT_statement;
```

```
CURSOR cursor_name
(parameter_list)
IS
   SELECT_statement;
```

```
CREATE OR REPLACE Function
FindCourse
   ( name_in IN varchar2 )
   RETURN number
IS
    cnumber number;
    CURSOR c1
    IS
       SELECT course_number
        from courses_tbl
        where course_name = name_in;

BEGIN
   open c1;
   fetch c1 into cnumber;
   if c1%notfound then
     cnumber := 9999;
   end if;
   close c1;
RETURN cnumber;
END;
```

Funções

- Sintaxe

```
CREATE [OR REPLACE] FUNCTION
function_name
   [ (parameter [,parameter]) ]
   RETURN return_datatype
IS | AS
   [declaration_section]
BEGIN
   executable_section
[EXCEPTION
   exception_section]
END [function_name];
```

Os parametros são opcionais. Existem três tipos de parâmetros:
IN,
OUT,
IN OUT
(Significado comum a outras linguagens)

```
/* A seguinte função fornece a tensão de um motor */
CREATE FUNCTION tensao_motor (Mot motores.motorID%TYPE)
RETURN motores.tensao%type IS
    tensao_ret     motores.tensao%type;
BEGIN
   SELECT tensao into tensao_ret FROM motores WHERE motorID = Mot;
   RETURN tensao_ret;
END tensao_motor;
```

Procedimento anónimo

```
declare
   tensao integer;
begin
   tensao:=tensao_motor('MRB01');
   dbms_output.put_line('tensao1=' || tensao );
   dbms_output.put_line ('chamar directamente:
tensao='||tensao_motor('MRB01'));
end;
```

RESULTADO:
tensao1=400
chamar directamente: tensao=400

Procedures (v2)

- Sintaxe

```
CREATE [OR REPLACE] PROCEDURE
procedure_name
   [ (parameter [,parameter]) ]
IS
   [declaration_section]
BEGIN
   executable_section
[EXCEPTION
   exception_section]
END [procedure_name];
```

```
CREATE PROCEDURE ler_nome_motores
(pot IN motores.potencia%type) IS
    nome_motor       motores.descricao%type;
    CURSOR     c1(pot   motores.potencia%type) IS
          SELECT descricao FROM motores
          WHERE potencia = pot
    BEGIN
       OPEN c1(pot);
       LOOP
          FETCH c1 INTO nome_motor;
          EXIT WHEN c1%NOTFOUND;
          DBMS_OUTPUT.PUT_LINE('Motor -> ' || nome_motor );
        END LOOP;
        CLOSE c1;
    END;
```

Os parametros são opcionais. Existem três tipos de parâmetros:
IN,
OUT,
IN OUT
(Significado comum a outras linguagens)

```
exec ler_nome_motores(4.5);
```

Triggers

- Tipos de triggers

Insert Triggers:
- BEFORE INSERT Trigger
- AFTER INSERT Trigger

Update Triggers:
- BEFORE UPDATE Trigger
- AFTER UPDATE Trigger

Delete Triggers:
- BEFORE DELETE Trigger
- AFTER DELETE Trigger

Drop Triggers:
- Drop a Trigger

Disable/Enable Triggers:
- Disable a Trigger
- Disable all Triggers on a table
- Enable a Trigger
- Enable all Triggers on a table

Before insert trigger

- Imaginemos a seguinte tabela:

```
CREATE TABLE orders
(    order_id          number(5),
     quantity          number(4),
     cost_per_item     number(6,2),
     total_cost        number(8,2),
     create_date       date,
     created_by        varchar2(10)
);
```

Avant l'insertion de données dans la table (e.g., pour limiter le nombre d'inscriptions d'élèves à une disicpline).

```
CREATE OR REPLACE TRIGGER orders_before_insert
BEFORE INSERT
   ON orders
   FOR EACH ROW
DECLARE
   v_username varchar2(10);
BEGIN
   -- Find username of person performing INSERT into table
   SELECT user INTO v_username
   FROM dual;
   -- Update create_date field to current system date
   :new.create_date := sysdate;
   -- Update created_by field to the username of the
   -- person performing the INSERT
   :new.created_by := v_username;
END;
```

After insert trigger

- Imaginemos a seguinte tabela:

```
CREATE TABLE orders
(   order_id        number(5),
    quantity        number(4),
    cost_per_item   number(6,2),
    total_cost      number(8,2)
);
```

```
CREATE OR REPLACE TRIGGER orders_after_insert
AFTER INSERT
   ON orders
   FOR EACH ROW
DECLARE
   v_username varchar2(10);
BEGIN
   -- Find username of person performing the INSERT into the table
   SELECT user INTO v_username
   FROM dual;
   -- Insert record into audit table
   INSERT INTO orders_audit
    ( order_id,
      quantity,
      cost_per_item,
      total_cost,
      username )
   VALUES
    ( :new.order_id,
      :new.quantity,
      :new.cost_per_item,
      :new.total_cost,
      v_username );
END;
```

Um trigger para vários “eventos”

```
CREATE TRIGGER  CLIENTES_CASCADE
   BEFORE   UPDATE  OR  DELETE  ON   CLIENTE   FOR   EACH
      ROW
DECLARE
      Contas_cli  INTEGER;
BEGIN
   IF UPDATING('REF_NO')  THEN
      /* propagar na conta_cliente a alteração de ‘ref_no’
      UPDATE  CONTA_CLIENTE
      SET CONTA_CLIENTE.REF_NO = :NEW.REF_NO
      WHERE CONTA_CLIENTE.REF_NO = :OLD.REF_NO;
  ELSIF  DELETING(‘REF_NO’)   THEN
      /* não deixa apagar cliente se tiver conta */
      SELECT COUNT(*)  INTO  Contas_cli
      FROM  CONTA_CLIENTE
      WHERE CONTA_CLIENTE.REF_NO =:OLD.REF_NO;
      IF Contas_cli > 0 THEN
          raise_application_error( -20501, 'Cliente possui
      contas!!!');
      END IF;
```

```
ELSIF  UPDATING(‘CONTA_NO’)
 /* não deixa alterar ‘conta_no’ se existir em conta_cliente */
    SELECT COUNT(*)  INTO  Contas_cli
    FROM  CONTA_CLIENTE
    WHERE CONTA_CLIENTE.REF_NO =:OLD.REF_NO;
    IF Contas_cli > 0 THEN
          raise_application_error(-20502, 'Cliente possui
contas!!!');
    END IF;
ELSIF  DELETING(‘CONTA_NO’)
  /* não deixa alterar ‘conta_no’ se existir em conta_cliente
*/
     SELECT COUNT(*)  INTO  Contas_cli
     DELETE
     FROM  CONTA_CLIENTE
     WHERE CONTA_CLIENTE.REF_NO =:OLD.REF_NO;
     IF Contas_cli > 0 THEN
          raise_application_error( -20502, 'Cliente possui
contas!!!');
      END IF;
END;
```