

APPLICATIONS WEB

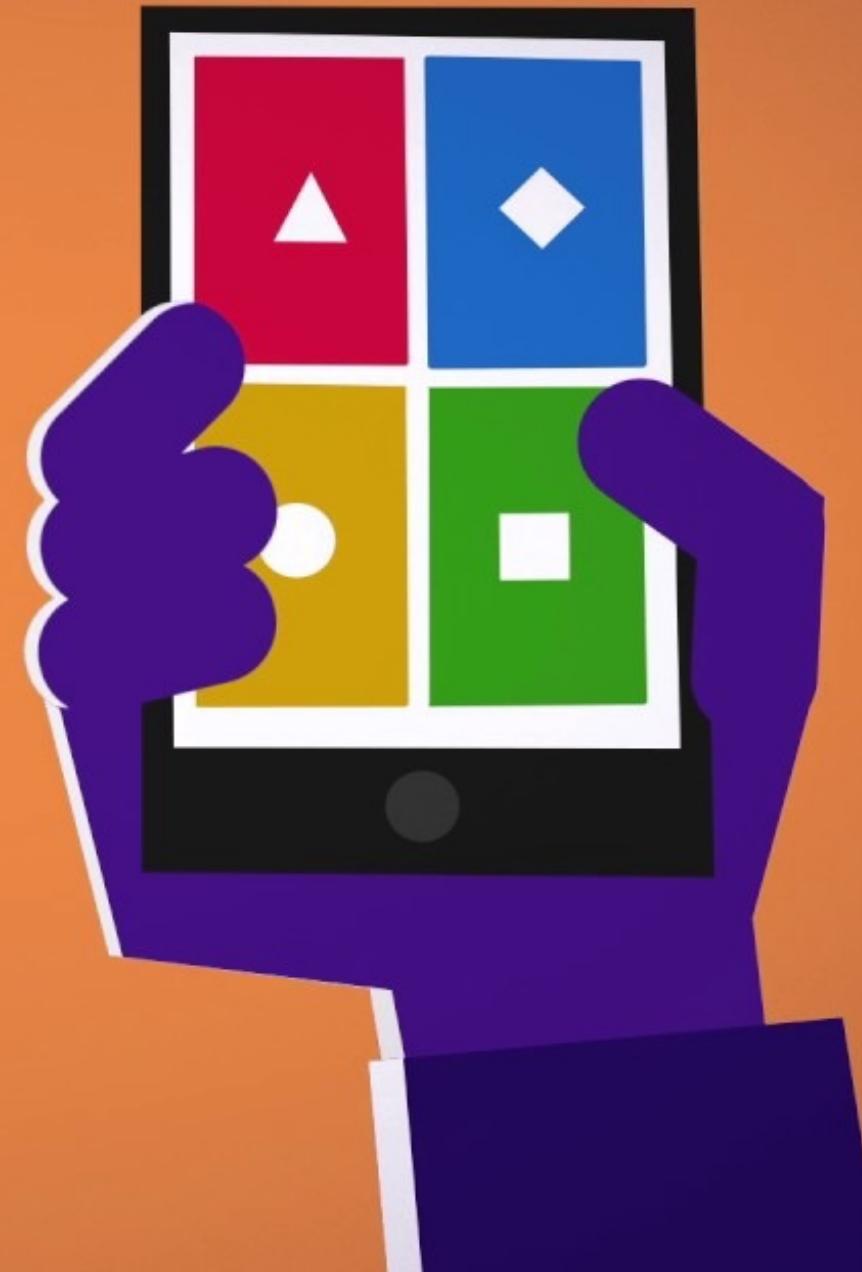


JEAN-FRANÇOIS BRODEUR

COURS 7 – LE LANGAGE SQL - 3

AGENDA DE LA SESSION

- Présences
- Kahoot du cours précédent (SQL-2)
- Révision du cours précédent
- Correction des Labos 1 et 2
- Présentation du matériel du cours 7
- Cours magistral sur SQL-3
- Laboratoire SQL – Meurtre et mystère
- Actualité et cyber sécurité



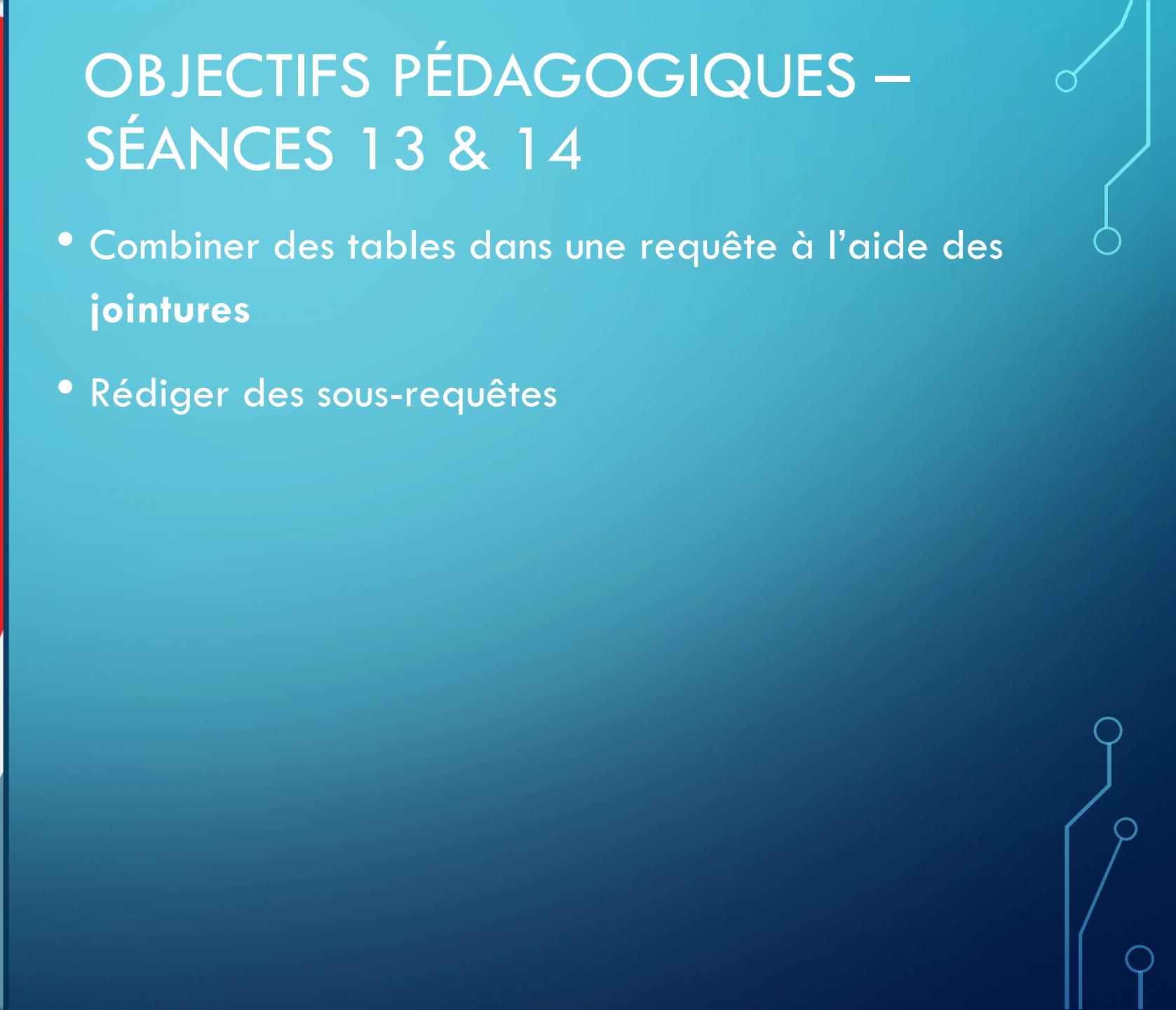


KAHOOT SQL - 2

- <https://create.kahoot.it/creator/98f24b18-9b71-4aed-98e1-7c5099d36c83>



OBJECTIFS PÉDAGOGIQUES – SÉANCES 13 & 14

- Combiner des tables dans une requête à l'aide des **jointures**
 - Rédiger des sous-requêtes
- 

OBJECTIFS PÉDAGOGIQUE – LEÇON 7

- Combiner des tables dans une requête à l'aide des **jointures**
- Rédiger des sous-requêtes

EXÉCUTER UN SCRIPT .SQL

- Ouvrir le fichier – charger le script « **Script création BD club vidéo.sql** »
- Positionner le curseur au début
- Faire exécuter
- Magique ! 😊

LES TABLES ET LEUR LIEN

EXEMPLE : FICHE D'UN CLIENT

Champ	Type	Contenu
Nom	texte	Stark
Prénom	texte	Ned
Date de naissance	date	17 avril 1959
Adresse	texte	Tour Nord
Ville	texte	Winterfell
Province	texte	The Nord
Téléphone	texte	N/A
Date d'enregistrement	date	1 septembre 2013

EXEMPLE : FICHE D'UN FILM

CHAMP	TYPE	CONTENU
Titre	texte	Bon cop, bad cop
Directeur	texte	Éric Canuel
Acteurs	texte	Patrick Huard, Colm Feore
Année	numérique	2006
Date d'enregistrement	date	20 octobre 2007



EMPRUNTS ?

- De quelles informations a-t-on besoin pour enregistrer un emprunt ?
 - Le client qui fait l'emprunt
 - Le film emprunté
 - Des données appartenant à l'emprunt :
 - Date d'emprunt
 - Date de retour (vide au moment de l'emprunt)
 - Certaines informations pourraient être calculées automatiquement ; par exemple, la date prévue de retour, etc.

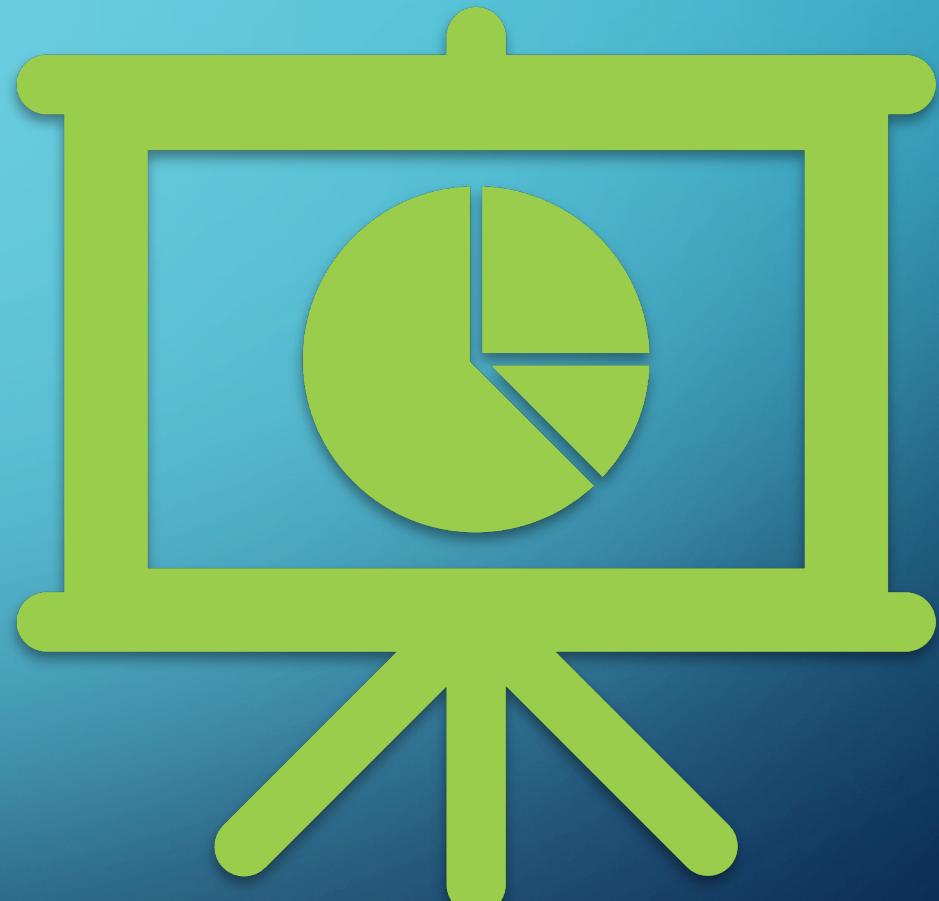
EXEMPLE : FICHE D'UN EMPRUNT

- On a besoin d'une façon simple d'identifier un enregistrement **de façon UNIQUE** (un film, un emprunteur).

Film	Emprunteur	Date d'emprunt	Date de retour
Bon cop, bad cop	Ned Stark	10 janvier 2014	[Vide]

LA CLÉ PRIMAIRE

- C'est une colonne (ou ensemble de colonnes) dont la valeur nous permet d'**identifier chaque enregistrement de façon unique (clé candidate)**.
 - Exemples :
 - Numéro d'assurance sociale (personnes)...
 - Courriel (moins certain... peut changer...)
 - ISBN (livres)...
 - Code de produit, etc.
 - En utilisant la clé primaire, **on peut faire référence** à un enregistrement au complet
 - → pas besoin de répéter des informations (nom, adresse, téléphone...)





RÈGLES POUR LA CLÉ PRIMAIRE

- Chaque valeur doit être **unique**
 - Je ne devrais pas pouvoir enregistrer un autre client avec le même numéro
- Elle **ne peut pas être nulle** (vide)
 - Je ne devrais pas pouvoir enregistrer un client qui n'a pas de numéro de client !
 - *(Attention : on peut trouver d'autres champs qui ne doivent pas être nulles, pas juste la clé primaire. Ex. le nom du client → car créer un client sans nom ne ferait pas de sens !)*

AJOUT D'UNE CLÉ PRIMAIRE

Champ	Type	Contenu
Numéro de client	numérique	123
Nom	texte	Stark
Prénom	texte	Ned
Date de naissance	date	17 avril 1959
Adresse	texte	Tour Nord
Ville	texte	Winterfell
Province	texte	Le Nord
Téléphone	texte	N/A
Date d'enregistrement	date	1 septembre 2012

EXEMPLE: FICHE D'UN FILM

Champ	Type	Contenu
Code du film	texte	BONC001
Titre	texte	Bon cop, bad cop
Directeur	texte	Éric Canuel
Acteurs	texte	Patrick Huard, Colm Feore
Année	numérique	2006
Date d'enregistrement	date	20 octobre 2007

EXAMPLE : FICHE D'UN EMPRUNT

- **Numéro d'emprunt** → clé primaire de l'emprunt !
- **Numéro de client** et **Code du film** font référence à des **enregistrements uniques** des tables Client et Film.
 - On les appelle **clés étrangères**.

CHAMP	TYPE	CONTENU
Numéro d'emprunt	Numérique	372099
Numéro de client	Numérique	123
Code du film	texte	BONC001
Date d'emprunt	date	10 janvier 2014
Date de retour	date	[Vide]

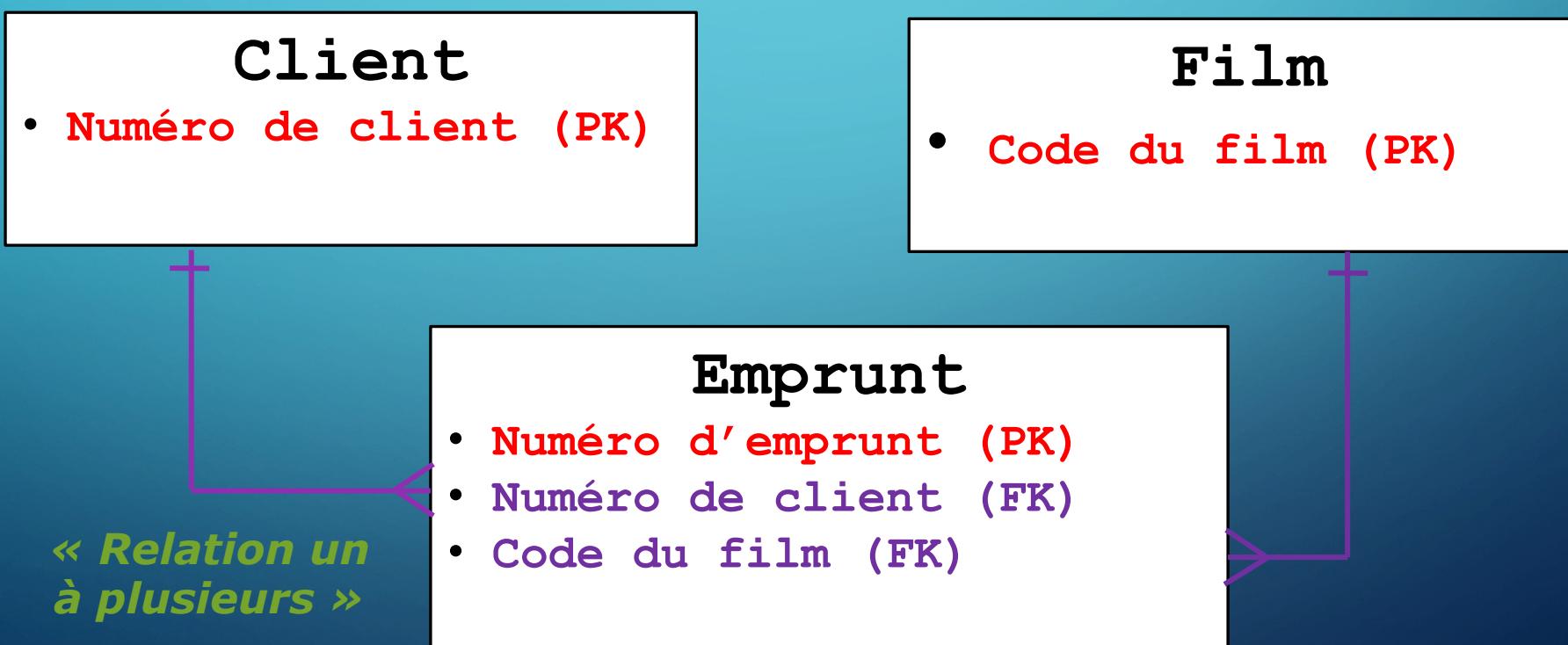
EXEMPLES D'EMPRUNTS

- Pour que ça ait du sens, il faut absolument que la base de données garantisse :
 - **L'existence des enregistrements « pointés » par Numéro de client et Code du film**
 - **L'unicité de leurs valeurs** (ex. **Code du film** doit être unique → il doit être une **clé primaire**)

Numéro d'emprunt	Numéro de client	Code du film	Date d'emprunt	Date de retour
372099	123	BONC001	10/01/2013	[Vide]
372100	501	STAR003	11/01/2013	13/01/2013
372101	400	HARR210	11/01/2013	15/01/2013

ASSOCIATIONS ENTRE LES TABLES

- Les bases de données modernes (dont Oracle) permettent de créer des **associations (relations)** entre les tables via leurs clés : PK: primary key





RELATIONS ENTRE LES TABLES

- Un client peut avoir **plusieurs** emprunts ... mais un emprunt appartient à **un** et un seul client.
- Un film peut apparaître dans **plusieurs** emprunts ... mais un emprunt fait référence à **un** et un seul film.
- Ce type de relation s'appelle « **relation un à plusieurs** ».
- La base de données va garantir ces relations :
 - Elle ne permettra pas qu'on rentre, par exemple, un numéro de client qui n'existe pas.
- Ces colonnes qui « pointent » vers la clé primaire d'une autre table s'appellent « **clés étrangères** ».

SQL, LES JOINTURES

- Les jointures en SQL permettent d'associer plusieurs tables dans une même requête.
- Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

- **INNER JOIN :**
 - jointure interne pour retourner les enregistrements quand la condition est vraie dans les 2 tables. C'est la jointure la plus commune.
- **CROSS JOIN :**
 - jointure croisée permettant de faire le produit cartésien de 2 tables. En d'autres mots, permet de joindre chaque ligne d'une table avec chacune ligne d'une seconde table. Attention, le nombre de résultats est en général très élevé.
- **LEFT JOIN (ou LEFT OUTER JOIN) :**
 - jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifiée dans l'autre table.

SQL, LES JOINTURES

LES JOINTURES

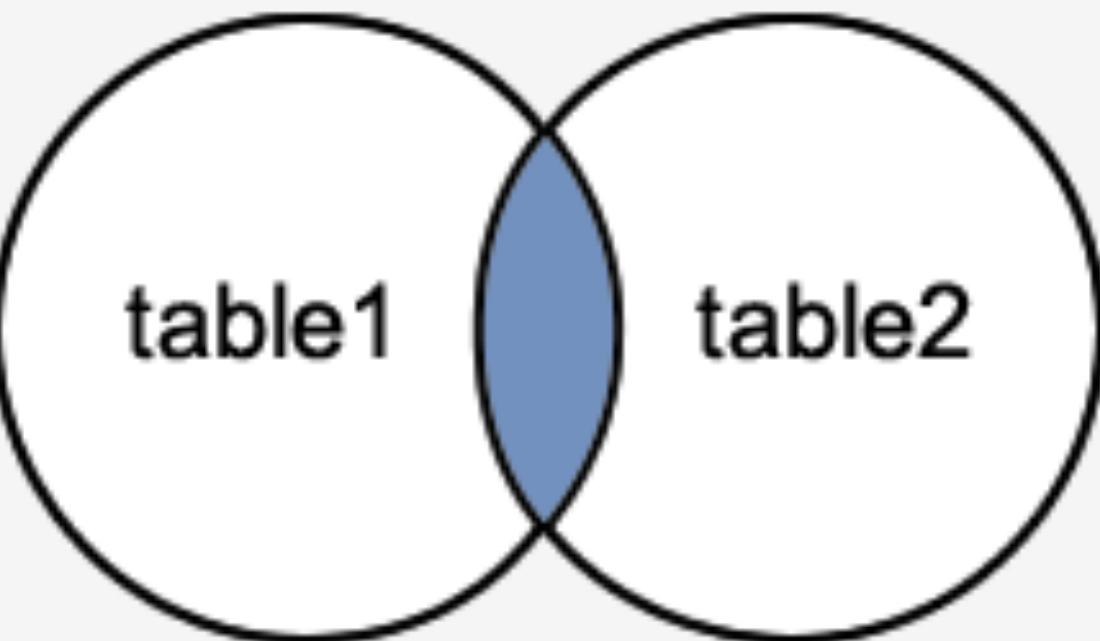
- **RIGHT JOIN (ou RIGHT OUTER JOIN) :**
 - jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifié dans l'autre table.
- **FULL JOIN (ou FULL OUTER JOIN) :**
 - jointure externe pour retourner les résultats quand la condition est vrai dans au moins une des 2 tables.
- **SELF JOIN :**
 - permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.

LES JOINTURES

```
SELECT table1.column, table2.column
FROM table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2 ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name = table2.column_name)] |
[CROSS JOIN table2];
```

LA SYNTAXE POUR UN INNER JOIN EST

```
SELECT colonne1,colonne2,...  
FROM table1 t1  
INNER JOIN table2 t2  
ON t1.colonne1 = t2.colonne2
```





SQL & LES VILLES ATTAQUÉES PAR GODZILLA

TABLES DE DÉMONSTRATION

Table A

City_name	Country
Tokyo	Japan
New York	USA
Fukuoka	Japan
Shanghai	China

Table B

City_name	Godzilla_attacks
Fukuoka	3
Nagoya	2
New York	3
Tokai	3
Tokyo	13
Yokohama	2

```
SELECT A.City_name, B.Godzilla_attacks  
FROM A  
JOIN B on A.City_name = B.City_name
```

- Voici comment le résultat de la requête est construit:
 - Pour toutes les rangées dans la tableA
 - Rechercher un appariement dans la tableB
 - Si trouvé
 - Afficher les colonnes des rangées de la tableA et de la tableB
 - Aucun trouvé
 - Rien affiché

COMMENT EST FAIT LA JOINTURE

VISUELLEMENT DEUX BOUCLES FOR

```
select A.City_name, Godzilla_attacks
from A
JOIN B on A.City_name = B.City_name
```

Table A

City_name	Country
Tokyo	Japan
New York	USA
Fukuoka	Japan
Shanghai	China

Table B

City_name	Godzilla_attacks
Fukuoka	3
Nagoya	2
New York	3
Tokai	3
Tokyo	13
Yokohama	2

Working Set

A.City_name	A.Country	B.City_name	B.Godzilla_attacks
Tokyo	Japan		
New York	USA		
Fukuoka	Japan		
Shanghai	China		
		Fukuoka	3
		Nagoya	2
		New York	3
		Tokai	3
		Tokyo	13
		Yokohama	2

JOINTURE

EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	20
5	206	20
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

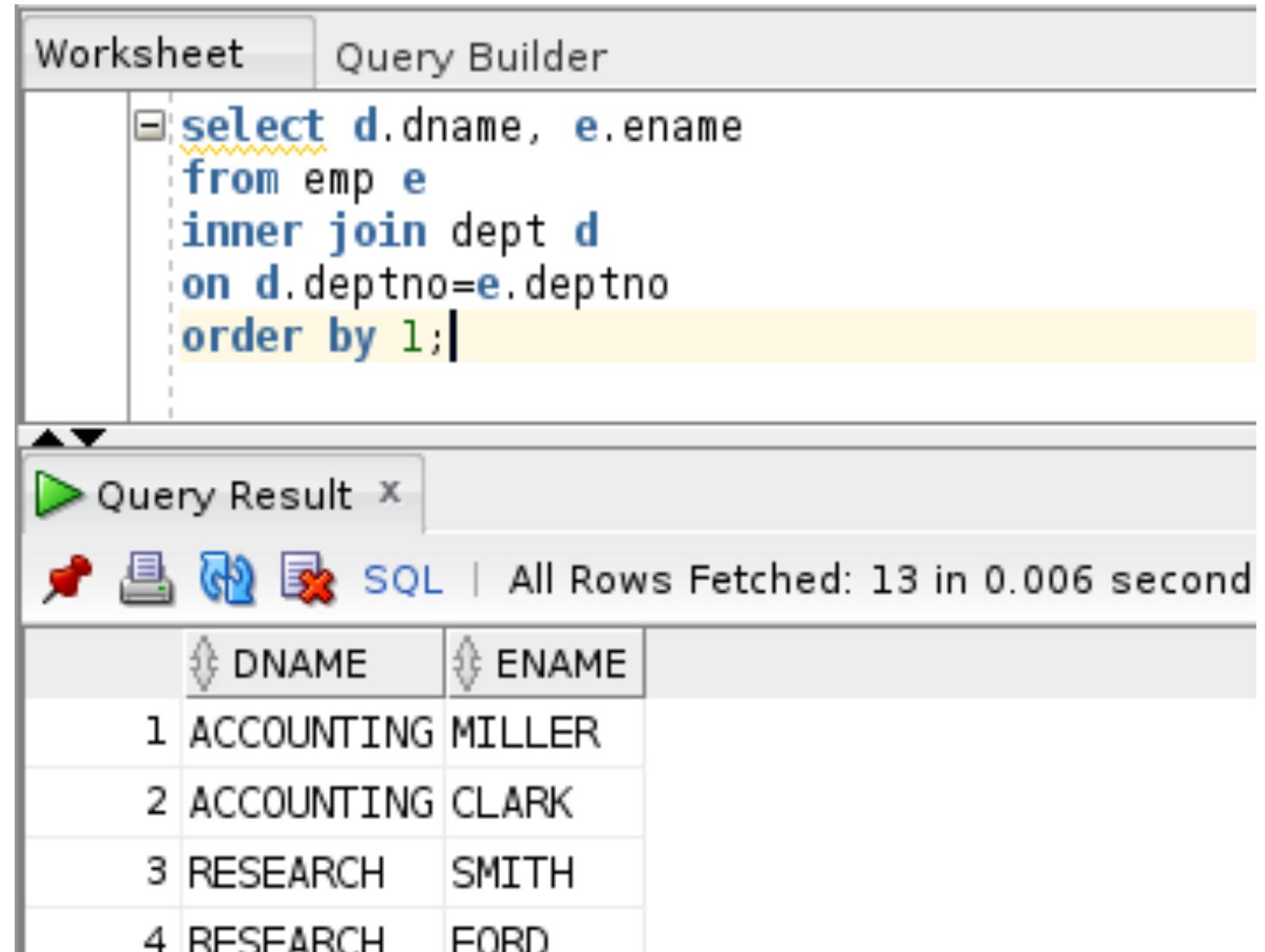
Equijoin

Foreign
Key

Primary
Key

SQL, INNER JOIN (ÉQUIJOIN)

- La liste des employés par description de département



The screenshot shows a database query tool interface. The top window is titled "Worksheet" and contains the following SQL code:

```
select d.dname, e.ename
from emp e
inner join dept d
on d.deptno=e.deptno
order by 1;
```

The bottom window is titled "Query Result" and displays the results of the query. The results are as follows:

	DNAME	ENAME
1	ACCOUNTING	MILLER
2	ACCOUNTING	CLARK
3	RESEARCH	SMITH
4	RESEARCH	FORD

JOINTURE SUR ELLE-MÊME

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst



MANAGER_ID in the WORKER table is equal to
EMPLOYEE_ID in the MANAGER table.

JOINTURE SUR ELLE-MÊME

- La liste des employés avec le nom de leur supérieur

Worksheet Query Builder

```
select e1.empno, e1.ename nom_employe, e2.ename nom_superieur
from emp e1
join emp e2
on e1.mgr=e2.empno;
```

Script Output x Query Result x

SQL | All Rows Fetched: 13 in 0.007 seconds

	EMPNO	NOM_EMPLOYE	NOM_SUPERIEUR
1	7902	FORD	JONES
2	7788	SCOTT	JONES
3	7844	TURNER	BLAKE
4	7499	ALLEN	BLAKE
5	7521	WARD	BLAKE
6	7900	JAMES	BLAKE
7	7654	MARTIN	BLAKE
8	7934	MILLER	CLARK
9	7876	ADAMS	SCOTT
10	7698	BLAKE	KING
11	7566	JONES	KING
12	7782	CLARK	KING
13	7369	SMITH	FORD

JOINTURE INÉGALE

- La liste des employés avec leur grade de paie

Worksheet Query Builder

```
select e.ename nam, e.sal salaire, s.grade "Échelon salarial"
from emp e
Join salgrade s
on e.sal
between s.losal and s.hisal;
```

Script Output x Query Result x

SQL | All Rows Fetched: 14 in 0.008 seconds

	NAM	SALAIRE	Échelon salarial
1	SMITH	800	1
2	JAMES	950	1
3	ADAMS	1100	1
4	WARD	1250	2
5	MARTIN	1250	2
6	MILLER	1300	2
7	TURNER	1500	3
8	ALLEN	1600	3
9	CLARK	2450	4
10	BLAKE	2850	4
11	JONES	2975	4
12	SCOTT	3000	4
13	FORD	3000	4
14	KING	5000	5

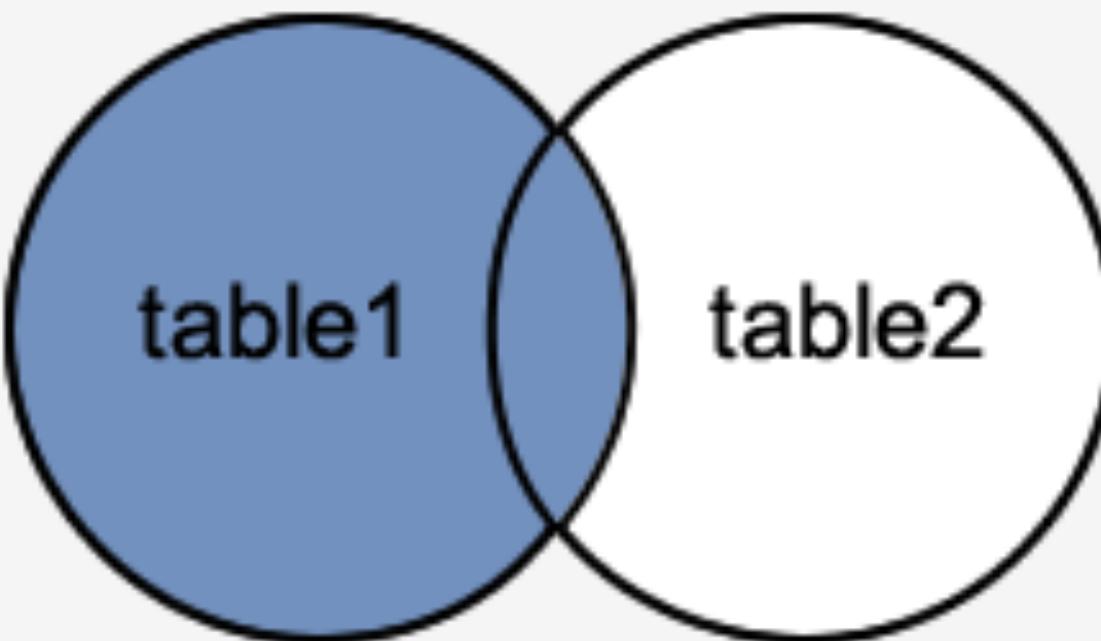
INNER JOIN / OUTER JOIN

- Quelle est la différence?
- Jointure INTERNE affiche les données qui sont présentes dans les deux tables et qui rencontrent la condition de jointure.
- Jointure EXTERNE va afficher toutes les rangées d'une table et les données de la seconde qui respectent la condition de jointure et affiche NULL si il n'y a pas de rangées dans la seconde table.
- En résumé, une jointure interne n'affiche que les rangées correspondantes dans les deux tables et une jointure externe affiche les correspondances et NULL pour les autres rangées.

SQL JOINTURES EXTERNES

- La syntaxe pour un **LEFT OUTER JOIN** est:

```
SELECT t1.colonne1, t2.colonne2, ...
FROM table1 t1
LEFT [OUTER] JOIN table2 t2
ON t1.colonne = t2.colonne;
```



```
SELECT A.City_name, B.Godzilla_attacks  
from A  
LEFT OUTER JOIN B on A.City_name = B.City_name
```

- Voici comment le résultat de la requête est construit:
 - Pour toutes les rangées dans la table1(A)
 - Afficher les colonnes de la table1
 - Rechercher un appariement dans la table2
 - Si trouvé
 - Afficher les colonnes des rangées de la table2 sur la même rangée de résultat que celles afficher précédemment de la table1
 - Aucun trouvé
 - Afficher null pour les colonnes des rangées de la table2

COMMENT EST FAIT LA JOINTURE

VISUELLEMENT

```
select city_name, godzilla_attacks  
from A  
LEFT JOIN B on A.City_name = B.City_name
```

Table A

City_name	Country
Tokyo	Japan
New York	USA
Fukuoka	Japan
Shanghai	China

Table B

City_name	Godzilla_attacks
Fukuoka	3
Nagoya	2
New York	3
Tokai	3
Tokyo	13
Yokohama	2

Working Set

A.City_name	A.Country	B.City_name	B.Godzilla_attacks
Tokyo	Japan		
New York	USA		
Fukuoka	Japan	Fukuoka	3
Shanghai	China		
		Nagoya	2
		New York	3
		Tokai	3
		Tokyo	13
		Yokohama	2

LEFT JOIN (LEFT OUTER)

- La liste des employés par département incluant les départements qui n'ont aucun employé en ordre descendant de numéro de département.
- La table de gauche est toujours la première déclarée dans la jointure

Worksheet Query Builder

```
select d.deptno departement, e.ename nom
from dept d
LEFT JOIN emp e
ON e.deptno=d.deptno
order by 1 desc;
```

Query Result All Rows Fetched: 14 in 0.011 seconds

DEPARTEMENT NOM

	DEPARTEMENT	NOM
1		40 (null)
2		30 WARD
3		30 ALLEN
4		30 BLAKE
5		30 JAMES
6		30 TURNER
7		30 MARTIN

UNE ENQUÊTE

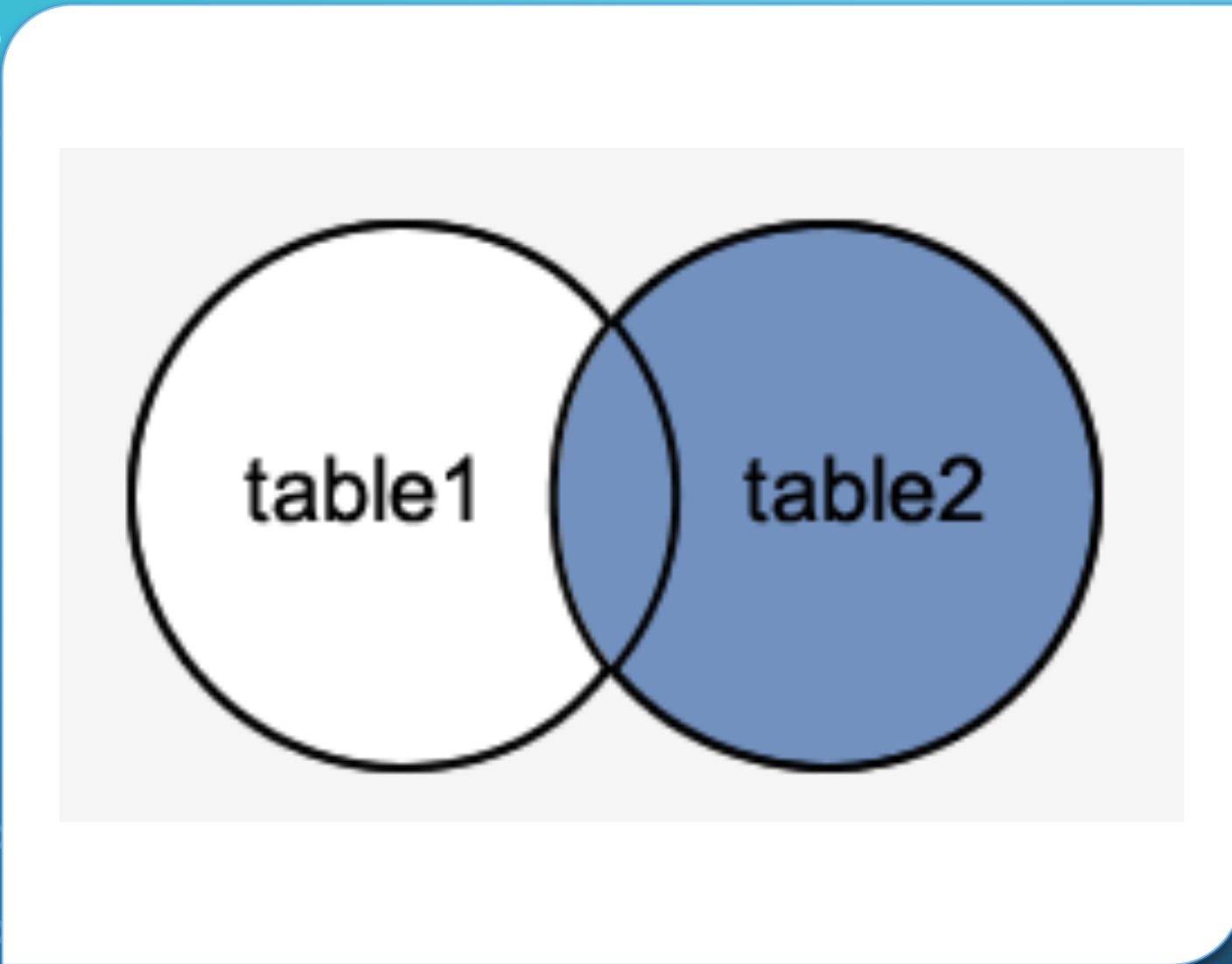
- <https://mystery.knightlab.com/>

SQL, JOINTURE ***GR2

10-3-2022

- La syntaxe pour un **RIGHT OUTER JOIN** est:

```
SELECT colonne1, colonne2,  
...  
FROM table1  
RIGHT [OUTER] JOIN table2  
ON table1.colonne =  
table2.colonne;
```



```
SELECT A.City_name, B.Godzilla_attacks  
from A  
RIGHT JOIN B on A.City_name = B.City_name
```

- Voici comment le résultat de la requête est construit:
 - Pour toutes les rangées dans la table2(B)
 - Afficher les colonnes de la table2
 - Rechercher un appariement dans la table1
 - Si trouvé
 - Afficher les colonnes des rangées de la table1 sur la même rangée de résultat que celles afficher précédemment de la table2
 - Aucun trouvé
 - Afficher null pour les colonnes des rangées de la table1

COMMENT EST FAIT LA JOINTURE

INNER JOIN (ÉQUIJOIN)

- Modifier KING pour qu'il n'ait pas de département assigné
 - UPDATE emp set deptno=null where ename='KING';
 - La liste des employés par département comprenant les employés qui n'ont pas de département associé

Worksheet Query Builder

```
select d.deptno departement, e.ename nom
from dept d
RIGHT JOIN emp e
ON e.deptno=d.deptno
order by 1 desc;
```

Query Result x

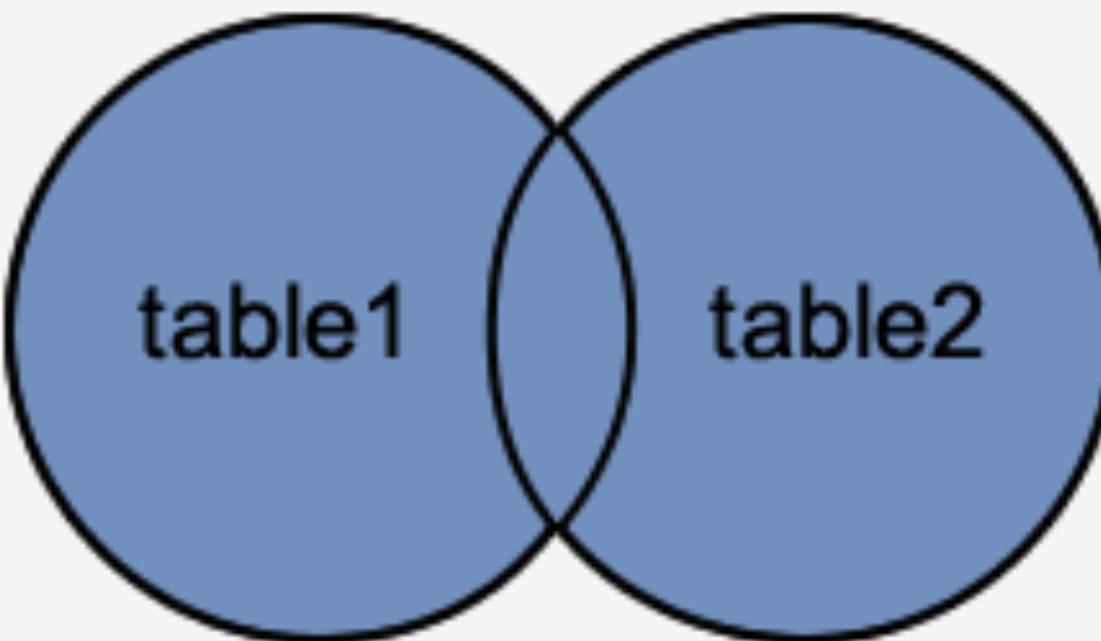
SQL | All Rows Fetched: 14 in 0.011 s

	DEPARTEMENT	NOM
1	(null)	KING
2	30	WARD
3	30	ALLEN
4	30	BLAKE
5	30	TURNER
6	30	JAMES
7	30	ADAMS
8	30	MARTIN
9	30	BLAKE
10	30	CLARK
11	30	KING
12	30	ADAMS
13	30	WARD
14	30	TURNER

JOINTURES EXTERNES COMPLÈTES

- La syntaxe pour un **FULL OUTER JOIN** est:

```
SELECT colonne1, colonne2,  
...  
FROM table1  
FULL [OUTER] JOIN table2  
ON table1.colonne =  
table2.colonne;
```



COMMENT EST FAIT LA JOINTURE

- Here's how the data is loaded:

For each record in table1

Afficher les données de la table

Chercher dans la table 2 selon la jointure

SI trouvé

Afficher les données de la table2

SINON

Afficher NULL

For each unmatched record in table2

Afficher les données de la table2

PRODUIT CARTÉSIEN

Worksheet Query Builder

```
select e.ename nam, e.sal salaire, d.dname
from emp e
cross Join dept d;
```

Script Output x Query Result x

SQL | Fetched 50 rows in 0.014 seconds

	NAM	SALAIRE	DNAME
1	SMITH	800	ACCOUNTING
2	ALLEN	1600	ACCOUNTING
3	WARD	1250	ACCOUNTING
4	JONES	2975	ACCOUNTING
5	MARTIN	1250	ACCOUNTING
6	BLAKE	2850	ACCOUNTING
7	CLARK	2450	ACCOUNTING
8	SCOTT	3000	ACCOUNTING
9	KING	5000	ACCOUNTING
10	TURNER	1500	ACCOUNTING
11	ADAMS	1100	ACCOUNTING
12	JAMES	950	ACCOUNTING

JOINTURES À TROIS TABLES

Worksheet Query Builder

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

Query Result x

SQL | All Rows Fetched: 19 in 0.3 seconds

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100 Seattle	Executive
2	101 Seattle	Executive
3	102 Seattle	Executive
4	103 Southlake	IT
5	104 Southlake	IT
6	107 Southlake	IT
7	124 South San Francisco	Shipping
8	141 South San Francisco	Shipping
9	142 South San Francisco	Shipping
10	143 South San Francisco	Shipping

Three way
Join



CROSS JOIN

- Cross Join, aussi nommé Produit cartésien
- Une jointure croisée est l'endroit où vous affichez les enregistrements des deux tables dans toutes les combinaisons possibles.
- La non-correspondance est effectuée sur les colonnes. Il affiche chaque enregistrement de table1 apparié à chaque enregistrement de table2.
- C'est ainsi que les données sont chargées :
 - Pour chaque enregistrement du tableau 1
 - Pour chaque enregistrement du tableau 2
 - Afficher les données des tableaux 1 et 2

ÉQUIJOINTURE ET NON-ÉQUIJOINTURE

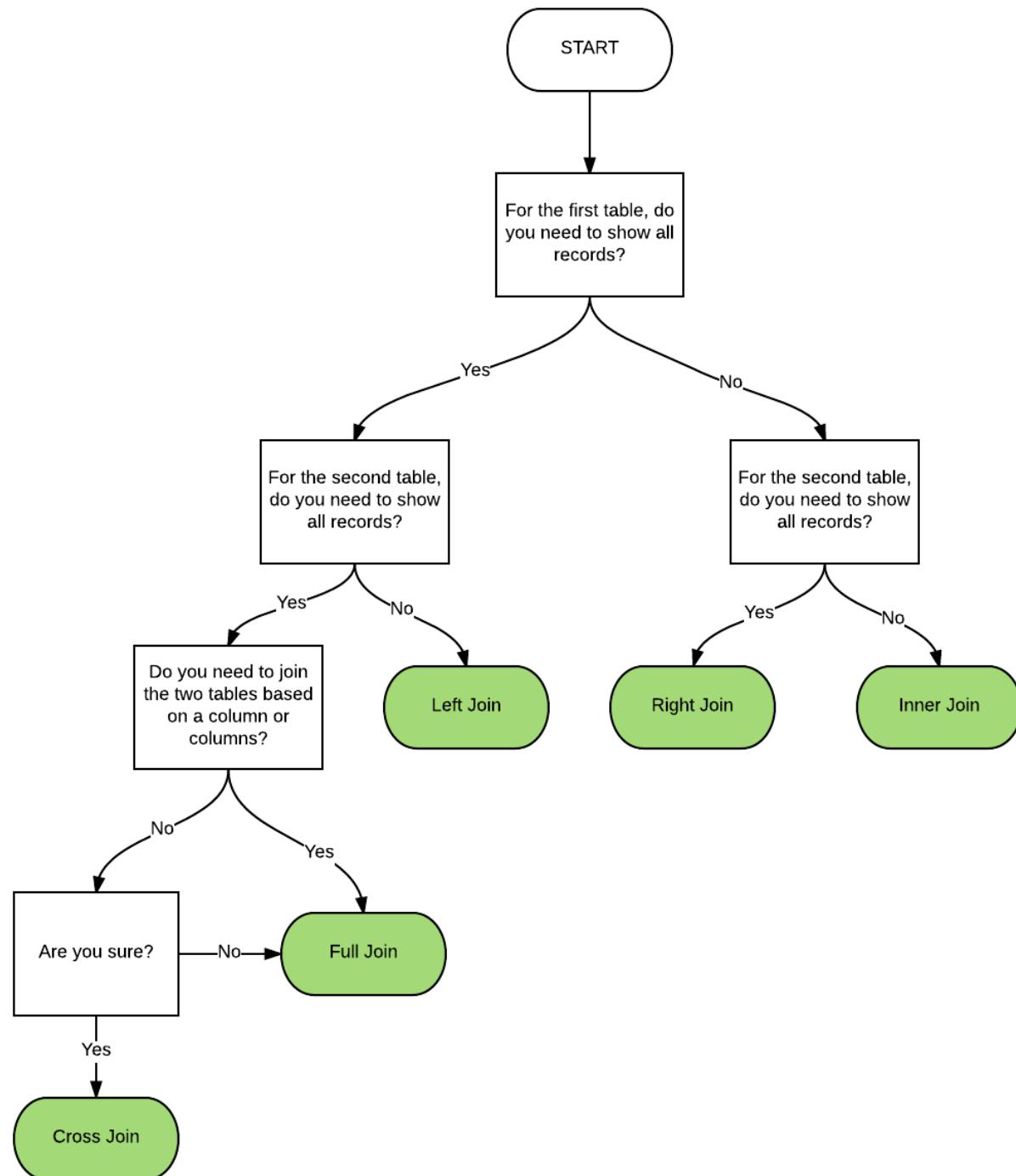
- Équijointure et non-équijointure ne sont pas des types de jointures, c'est plutôt un qualifiant du type jointure. Ainsi une jointure externe ou interne peut-être équijointure ou non.
- Une Équijointure est une jointure où l'appariement des colonnes est faite avec le signe d'égalité.

```
SELECT columns  
FROM table1  
INNER JOIN table2 ON table1.start_date = table2_effective_date;
```

- Dans une non-équijointure l'opérateur d'appariement de colonnes est tout autre signe que l'égalité (\geq , \leq , $>$, $<$ or \neq).

```
SELECT columns  
FROM table1  
INNER JOIN table2 ON table1.start_date  $\geq$  table2_effective_date;
```

QUEL JOINT CHOISIR



COMPARAISON DES TYPES DE JOINTURES

Critères	Inner	Left	Right	Full	Cross	Natural
Tous les enregistrements de la table 1?	Non	Oui	Non	Oui	Oui	Non
Tous les enregistrements de la table 2?	Non	Non	Oui	Oui	Oui	Non
Permettre de colonnes de nom différent?	Oui	Oui	Oui	Oui	N/A	Non

WHAT IF I TOLD YOU...

- Que vous pouvez faire une requête dans une autre requête ???

```
SELECT * FROM (  
    SELECT * FROM EMP  
    ORDER BY SAL DESC)
```

- Query-ception !!!





SOUS-REQUÊTE

- Une sous requête peut être utilisé:
 - À la place d'une colonne
 - À la place d'une expression dans le WHERE
 - Au lieu d'une table dans la clause FROM
 - Dans la section de HAVING

SOUS-REQUÊTE DE WHERE

- La sous-requête s'exécute en premier et retourne sa réponse à la requête principale

```
SELECT select_list  
FROM table  
WHERE expr operator
```

```
(SELECT select_list  
FROM table);
```

- L'opérateur peut être un =, LIKE, IN...

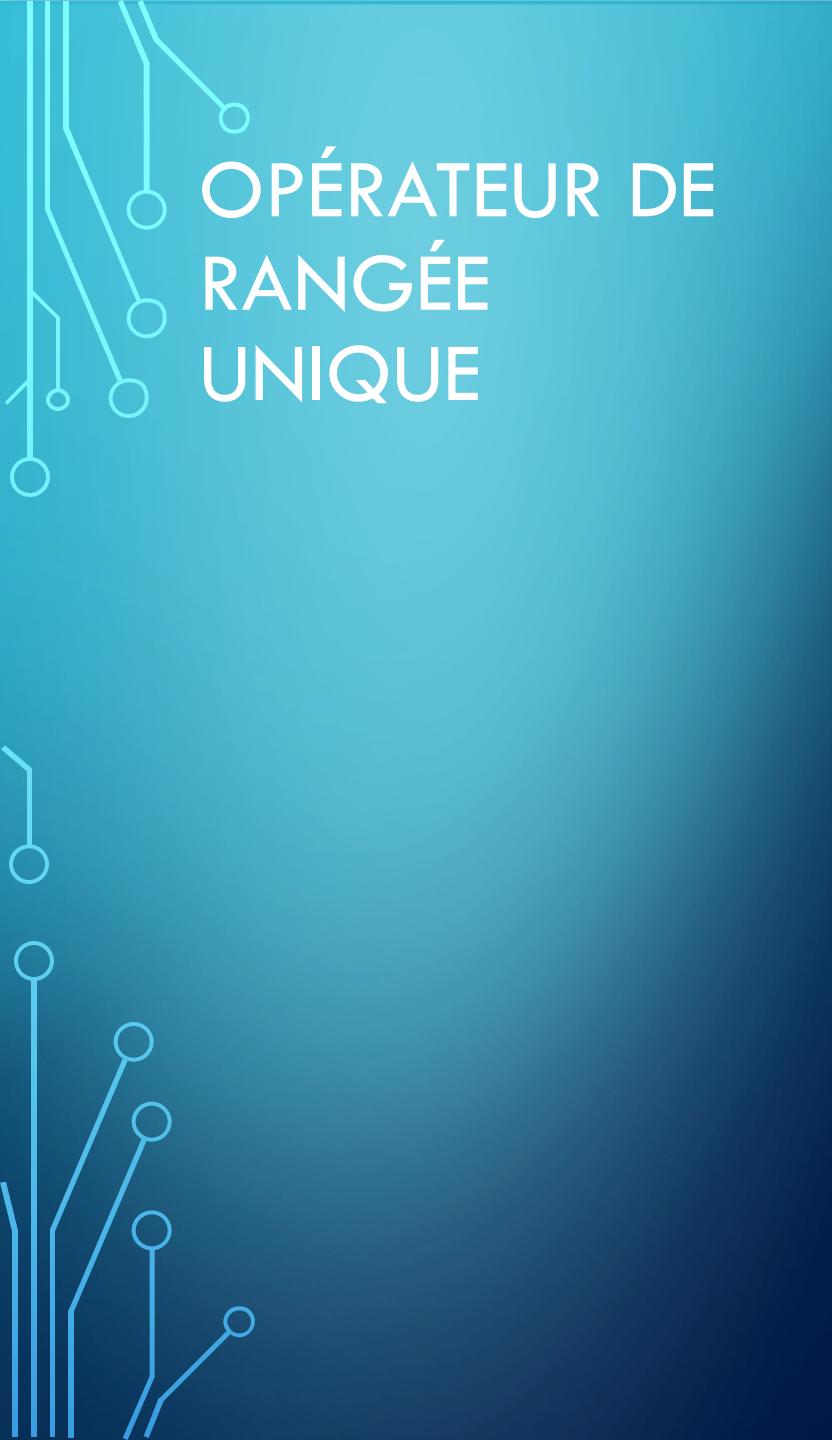
TYPE DE SOUS- REQUÊTE

- Single-row subquery



- Multiple-row subquery





OPÉRATEUR DE RANGÉE UNIQUE

Single-row Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

SOUS- REQUÊTES

- La liste de tous les employés embauchés après CLARK.

Worksheet Query Builder

```
select ename nam, hiredate
  from emp
 where hiredate > (select hiredate from emp where ename='CLARK')
  order by hiredate;
```

Script Output x Query Result x

SQL | All Rows Fetched: 8 in 0.023 seconds

	NAM	HIREDATE
1	TURNER	81-09-08
2	MARTIN	81-09-28
3	KING	81-11-17
4	JAMES	81-12-03
5	FORD	81-12-03
6	MILLER	82-01-23
7	SCOTT	87-04-19
8	ADAMS	87-05-23

OPÉRATEUR DE RANGÉE UNIQUE

Worksheet | Query Builder

```
SELECT employee_id, last_name
  FROM employees
 WHERE salary =  
      (SELECT MIN(salary)
       FROM employees
      GROUP BY department_id);
```

Equality operator expects only one value!

Query Result | Executing:SELECT employee_id, last_nameFROM employeesWHERE salary = (SELECT MIN(salary)) FROM employees GROUP BY

ORA-01427: single-row subquery returns more than one row
01427. 00000 - "single-row subquery returns more than one row"
*Cause:
*Action:

OPÉRATEUR DE RANGÉES MULTIPLE

Multiple-row
subquery

Returns
more than
one row

Use a
multiple-row
operator

OPÉRATEUR DE RANGÉES MULTIPLE

Multiple-row operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. This returns TRUE if at least one element exists in the result set of the subquery for which the relation is TRUE.
ALL	Must be preceded by =, !=, >, <, <=, >=. This returns TRUE if the relation is TRUE for all elements in the result set of the subquery.

SOUS-REQUÊTES

Un autre exemple : le mot clé IN

« *Donne-moi les employés qui appartiennent à un département situé à Chicago ou Dallas* »

- On pourrait faire d'abord :

SELECT DEPTNO

FROM DEPT

WHERE LOC = 'CHICAGO' OR LOC = 'DALLAS'

- Puis :

SELECT * FROM EMP WHERE DEPTNO IN (20, 30)

- ... ouf c'est plate !!!!

SOUS- REQUÊTES

- C'est beaucoup plus cool de dire :

```
SELECT *
```

```
FROM EMP
```

```
WHERE DEPTNO IN (
```

```
    SELECT DEPTNO FROM DEPT
```

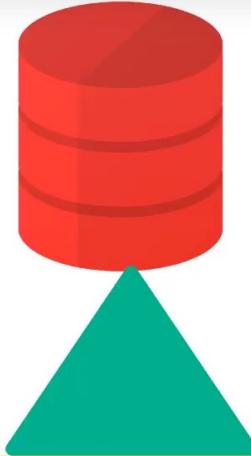
```
    WHERE LOC = 'CHICAGO' OR LOC = 'DALLAS' );
```

- Ici, la sous-requête (ou requête imbriquée) s'exécute en premier. Elle obtient les valeurs et la requête devient ainsi :

```
SELECT * FROM EMP WHERE DEPTNO IN  
(20,30);
```

SOUS- REQUÊTES

Oracle
Server



```
SELECT last_name, salary, department_id  
FROM employees  
WHERE salary IN (2500, 4200, 4400, 6000, 7000,  
8300, 8600, 17000) ;
```



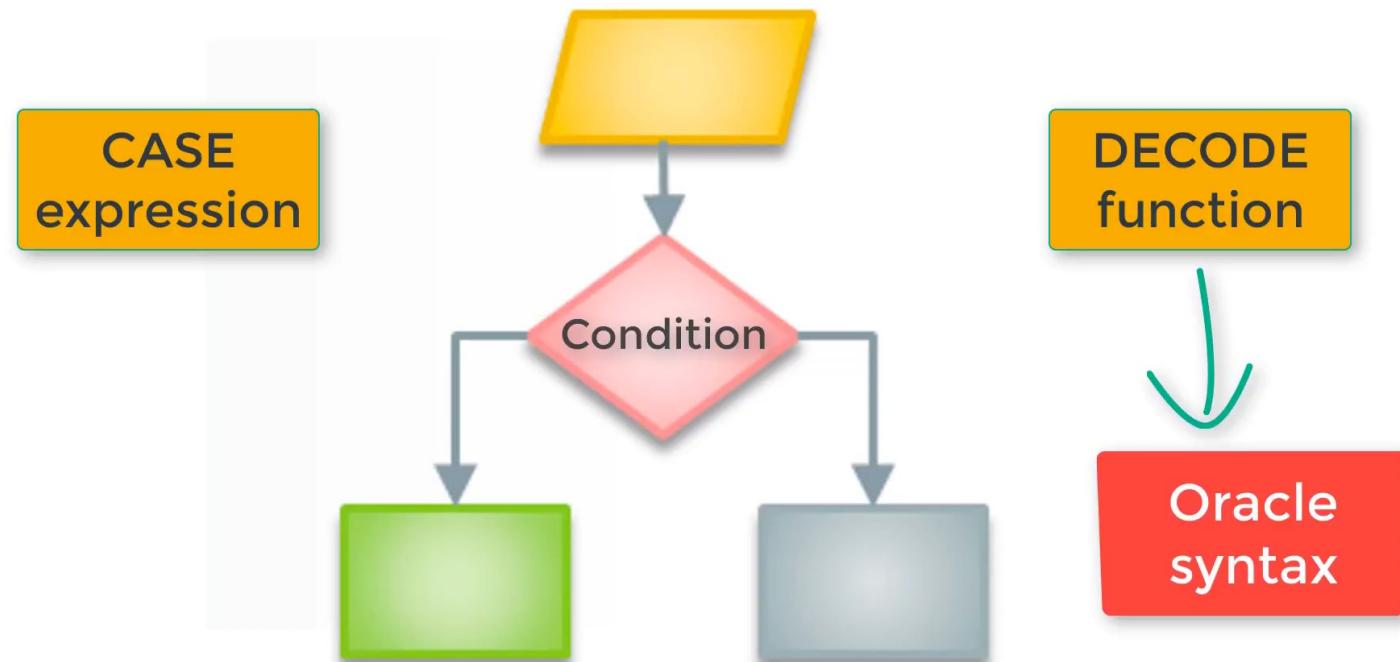
LE CASE?

LA SYNTAXE DU CASE

```
CASE expr WHEN comparison_expr1 THEN return_expr1
  [ WHEN comparison_expr2 THEN return_expr2
    WHEN comparison_exprn THEN return_exprn
    ELSE else_expr ]
END
```

LES EXPRESSIONS CONDITIONNELLES

- Pour ce cours seul la function CASE est utilisé



EXPRESSION CASE

Prend 2 formes :

1) Avec des valeurs discrètes:

```
SELECT ENAME, DEPTNO,  
CASE DEPTNO  
WHEN 10 THEN 'Comptabilité'  
WHEN 20 THEN 'Recherche'  
ELSE 'Autre'  
END AS NOM_DEPT  
FROM EMP  
ORDER BY DEPTNO;
```

On peut aussi établir des conditions plus complexes :

```
SELECT SAL,  
CASE  
WHEN SAL < 1000 THEN 'Bas'  
WHEN SAL >= 3000 THEN 'Haut'  
ELSE 'Moyen'  
END AS DESCRIPTION_SAL  
FROM EMP  
ORDER BY SAL;
```

EXEMPLE DE CASE

Worksheet Query Builder

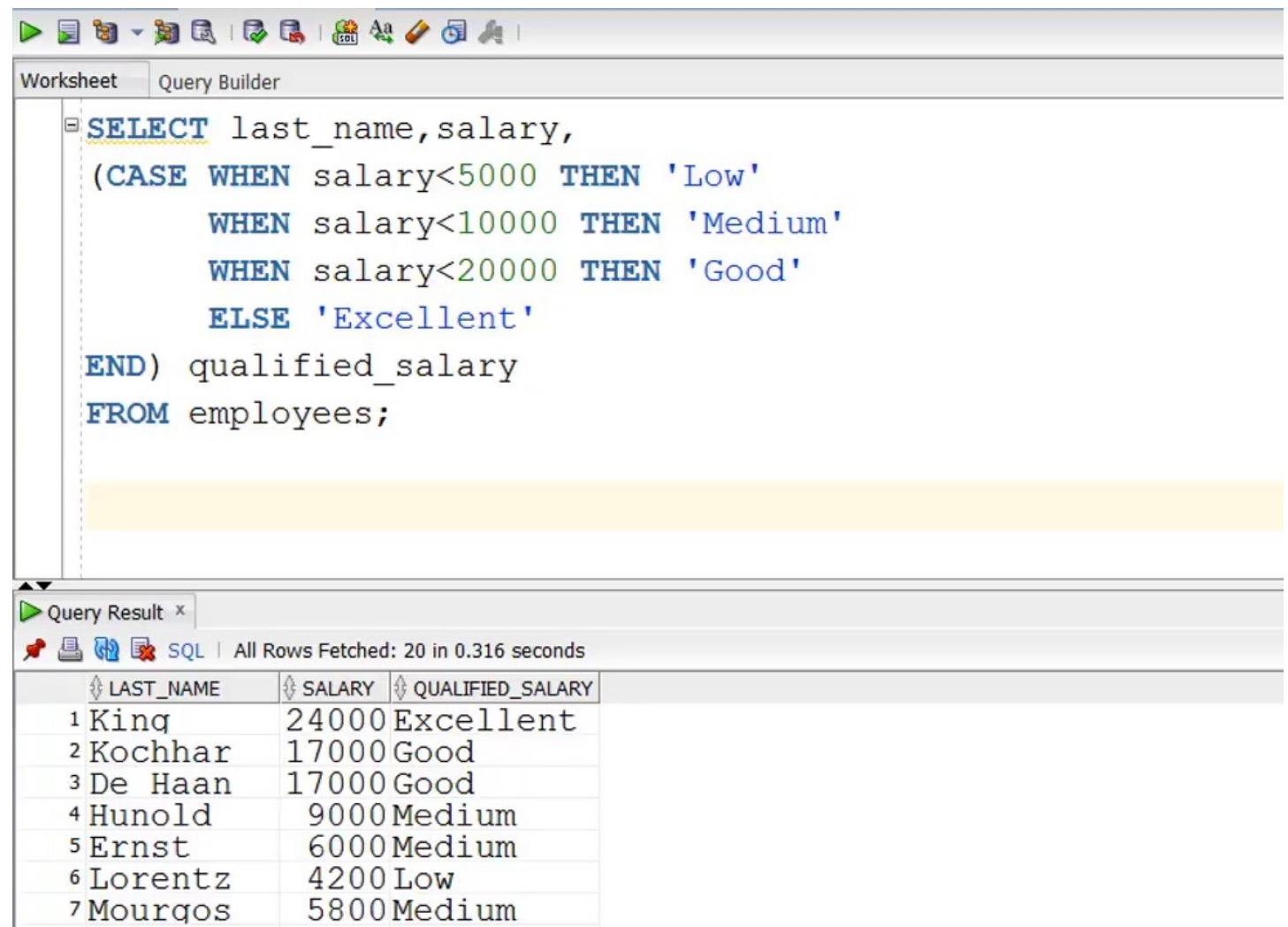
```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
                     WHEN 'ST_CLERK' THEN 1.15*salary
                     WHEN 'SA REP' THEN 1.20*salary
                     ELSE salary END      "REVISED_SALARY"
FROM employees;
```

Query Result x

SQL | All Rows Fetched: 20 in 0.311 seconds

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
1	King	AD PRES	24000	24000
2	Kochhar	AD VP	17000	17000
3	De Haan	AD VP	17000	17000
4	Hunold	IT PROG	9000	9900
5	Ernst	IT PROG	6000	6600
6	Lorentz	IT PROG	4200	4620
7	Mourgos	ST MAN	5800	5800
8	Rajs	ST CLERK	3500	4025
9	Davies	ST CLERK	3100	3565

EXEMPLE DE CASE



The screenshot shows a database query tool interface with two main panes. The top pane is a 'Worksheet' showing the SQL query:

```
SELECT last_name, salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```

The bottom pane is a 'Query Result' table showing the output:

LAST_NAME	SALARY	QUALIFIED_SALARY
1 King	24000	Excellent
2 Kochhar	17000	Good
3 De Haan	17000	Good
4 Hunold	9000	Medium
5 Ernst	6000	Medium
6 Lorentz	4200	Low
7 Mourgos	5800	Medium

LES VUES

- Les vues sont simplement des tables virtuelles qui sont formées en fonction des résultats obtenus à partir d'une requête SQL comprenant ou non des jointures.
- il s'agit d'une table logique, ce qui signifie que les vues n'existent pas dans la BD, elles ne nécessitent donc aucun espace dans la base de données.
- Les vues peuvent être créées à l'aide des tables de la base de données, nous pouvons utiliser une ou plusieurs tables pour créer les vues en fonction de la condition dans laquelle nous voulons créer les vues

LES VUES

Syntaxe pour créer une vue dans SQL :

```
CREATE VIEW nom_de_la_vue AS  
SELECT colonne_1, colonne _2, ..... colonne _n  
FROM table_1, table_2, .....table_n  
WHERE condition;
```

Pour afficher les données présentes dans la vue, utilisez l'instruction SELECT

```
SELECT * FROM nom_de_la_vue;
```

LES VUES

- **Utilisations des vues dans SQL :**
- **Les vues dans SQL sont utilisées pour réduire l'utilisation des tables.**
- **Une base de données appropriée doit contenir des vues pour:**
 - Restreindre l'accès aux données
 - Stocker de requêtes complexes
 - Masquer la complexité des données lors de l'utilisation de plusieurs tables.
 - Renommer des colonnes

LES VUES

- Les vues dans SQL permettent à l'utilisateur de sélectionner les informations requises à partir de plusieurs tables sans utiliser l'opération de jointure.
- Les vues dans SQL fournissent une sécurité supplémentaire en restreignant l'accès.
- Les vues masquent la complexité qui provient de l'utilisation de plusieurs tables.
- Les vues ont la capacité de stocker les requêtes volumineux.
- Renommer les colonnes signifie que sans affecter la table d'origine utilisée pour créer une vue, nous pouvons renommer les colonnes en vue, ce qui peut être fait en spécifiant dans l'instruction SELECT.
- Nous pouvons créer différentes vues en utilisant la même table pour différentes raisons ou pour différents utilisateurs.
- Les vues dans SQL utilisent ZERO Space dans la base de données, de sorte que notre base de données est libre d'espace, elle ne copie pas les données dans la base de données.
- Il peut facilement mettre à jour, supprimer les lignes de la vue qui est appelée une table virtuelle.
- Les vues aident à améliorer les performances et à partitionner les données.



L'ACTUALITÉ ET LA SÉCURITÉ

- https://www.theregister.com/2022/03/08/azure_awarp_flaw/
- https://www.theregister.com/2022/03/09/china_apt41_mandiant_usaherds/