SQL Intermedio: Datos Agregados, Joins y Operadores de Conjuntos

Datos Agregados y Ordenamiento

Qué son funciones de grupo?

- Las funciones de grupo operan sobre conjuntos de tuplas para dar un resultado por grupo.

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
	7000
10	4400

...

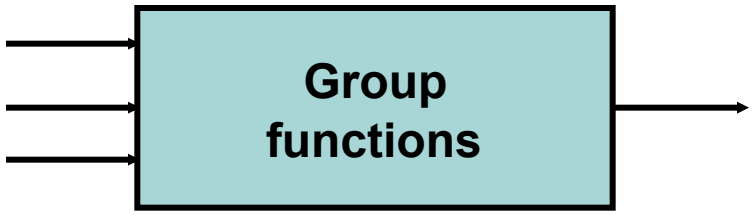
20 rows selected.

Maximum salary in EMPLOYEES table

MAX(SALARY)
24000

Tipos de funciones de grupos

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE



Sintaxis de funciones de grupos

```
SELECT      [column,] group function(column), ...
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

Usando las funciones AVG y SUM

- Puede usar AVG y SUM para datos numéricos

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

Usando the MIN and MAX Functions

- Puede usar MIN y MAX para tipos de datos numéricos y fechas.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM       employees;
```

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

Usando la función COUNT

- `COUNT(*)` retorna el número de filas de una tabla

1

```
SELECT  COUNT(*)
FROM    employees
WHERE   department_id = 50;
```

COUNT(*)
5

- `COUNT(expr)` el número de filas con valores no nulos para la expresión *expr*:

2

```
SELECT  COUNT(commission_pct)
FROM    employees
WHERE   department_id = 80;
```

COUNT(COMMISSION_PCT)
3

Usando el DISTINCT

- COUNT(DISTINCT *expr*) retorna el número de valores distintos no nulos de *expr*.
- Para mostrar el número de departamentos distintos en la tabla EMPLOYEES:

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

COUNT(DISTINCTDEPARTMENT_ID)
7

Funciones de grupos y valores nulos

- Las funciones de grupos ignoran los valores nulos en una columna:

1

```
SELECT AVG(commission_pct)
FROM   employees;
```

AVG(COMMISSION_PCT)
.2125

- La función NVL fuerza a las funciones de grupo a incluir valores nulos:

2

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

AVG(NVL(COMMISSION_PCT,0))
.0425

Creando grupos de datos

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
80	10500
80	8600
80	11000
90	24000
90	17000

...

20 rows selected.

4400

9500

3500

6400

10033

Average salary in EMPLOYEES table for each department

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

Creando grupos de datos: sintaxis GROUP BY

```
SELECT     column, group_function(column)
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

- Se puede particionar las tuplas de una tabla con la cláusula `GROUP BY`.

Usando la cláusula GROUP BY

- Todas las columnas en la lista `SELECT` que no son funciones de grupo deben estar en la cláusula GROUP BY

```
SELECT    department id, AVG(salary)
FROM      employees
GROUP BY department_id ;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

8 rows selected.

Usando la cláusula GROUP BY

- La columna `GROUP BY` no tiene por qué estar en la lista `SELECT`.

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```

AVG(SALARY)
4400
9500
3500
6400
10033.3333
19333.3333
10150
7000

Agrupando por más de una columna

EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
90	AD_PRES	24000
90	AD_VP	17000
90	AD_VP	17000
60	IT_PROG	9000
60	IT_PROG	6000
60	IT_PROG	4200
50	ST_MAN	5800
50	ST_CLERK	3500
50	ST_CLERK	3100
50	ST_CLERK	2600
50	ST_CLERK	2500
80	SA_MAN	10500
80	SA_REP	11000
80	SA_REP	8600

...

20	MK_REP	6000
110	AC_MGR	12000
110	AC_ACCOUNT	8300

20 rows selected.

Add the salaries in the `EMPLOYEES` table for each job, grouped by department

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

Usando la cláusula GROUP BY sobre múltiples columnas

```
SELECT    department_id dept_id, job_id, SUM(salary)
FROM      employees
GROUP BY department_id, job_id ;
```

DEPT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

Queries no válidos que usan funciones de grupo

- Toda columna o expresión en la lista `SELECT` que no es una función de grupos debe estar en la cláusula GROUP BY

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

```
SELECT department_id, COUNT(last_name)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Columna faltante en la cláusula `GROUP BY`

Queries no válidos que usan funciones de grupo

- No se puede usar el `WHERE` para restringir grupos.
- Se puede usar el `HAVING` para restringir grupos.
- No puede usar funciones de grupo en el WHERE

```
SELECT   department_id, AVG(salary)
FROM     employees
WHERE    AVG(salary) > 8000
GROUP BY department_id;
```

```
WHERE  AVG(salary) > 8000
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

No se puede usar el WHERE para restringir grupos

Restringiendo los resultados de grupos

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
...	
20	6000
110	12000
110	8300

20 rows selected.

The maximum salary per department when it is greater than $10,000

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

Restringiendo los resultados de grupos utilizando el HAVING

- When you use the `HAVING` clause, the Oracle server restricts groups as follows:
  1. Rows are grouped.
  2. The group function is applied.
  3. Groups matching the `HAVING` clause are displayed.

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

Usando la cláusula HAVING

```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY  department_id
HAVING    MAX(salary)>10000 ;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

Usando la cláusula HAVING

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

Funciones de grupo anidadas

- Máximo promedio de salario:

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

MAX(AVG(SALARY))
19333.3333

Usando la cláusula ORDER BY

- Ordenar tuplas con `ORDER BY`:
  - `ASC`: ascending order, default
  - `DESC`: descending order
- The `ORDER BY` clause comes last in the `SELECT` statement:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91

...

20 rows selected.

Ordenando

- Ordenando en orden descendente:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date DESC ;
```

1

- Ordenando por alias de columna:

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal ;
```

2

- Ordenando por múltiples columnas:

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```

3

Joins

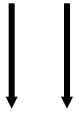
Obteniendo datos de múltiples tablas

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700



EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

Tipos de Joins

- Joins que siguen estándar SQL99 incluyen lo siguiente:
  - Productos cartesianos
  - `NATURAL JOIN` con cláusula `USING`
  - Full (left, right) outer joins
  - Condiciones de outer join arbitrarias

Joining de tablas usando sintaxis SQL 1999

- Usando el join para consultar datos de más de una tabla:

```
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];
```

Creando Natural Join

- La cláusula `NATURAL JOIN` está basada en las columnas de las tablas que tienen el mismo nombre.
- Selecciona las tuplas de las tablas que tienen los mismos valores en las columnas del mismo nombre.
- Si las columnas con el mismo nombre tienen diferente tipo, ocurre un error.

Recuperando Registros con Natural Join

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

Creando Joins con la cláusula USING

- Si varias columnas tienen el mismo nombre pero sus tipos no coinciden, la cláusula `NATURAL JOIN` puede ser modificada con la cláusula `USING` para especificar las columnas que deben ser usadas en un equijoin.
- Use la cláusula `USING` para hacer “match” de una columna cuando existe más de una con nombres iguales.
- No use el nombre de la tabla o alias en las columnas referenciadas.
- La cláusula `NATURAL JOIN` y `USING` son mutuamente exclusivas.

Join de nombres de columnas

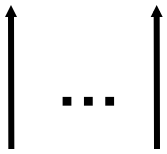
EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales



...

Foreign key **Primary key**

Recuperando registros con la clúsula USING

```
SELECT employees.employee_id, employees.last_name,
       departments.location_id, department_id
FROM   employees JOIN departments
USING (department id) ;
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
200	Whalen	1700	10
201	Hartstein	1800	20
202	Fay	1800	20
124	Mourgos	1500	50
141	Rajs	1500	50
142	Davies	1500	50
144	Vargas	1500	50
143	Matos	1500	50

...

19 rows selected.

Calificando nombres de columnas ambiguas

- Use prefijos de tablas que califican columnas con el mismo nombre.
- Use prefijos de tablas para mejorar desempeño.
- Use alias de columnas para distinguir columnas que están en distintas tablas pero tienen el mismo nombre
- No use alias sobre columnas que están en la cláusula `USING` y listadas en otra parte de la instrucción SQL.

Usando alias de tablas

- Uso de alias de tablas para simplificar consultas.
- Uso de alias de tablas para mejorar desempeño.

```
SELECT e.employee_id, e.last_name,
       d.location_id, department_id
FROM   employees e JOIN departments d
USING (department_id) ;
```

Creando Joins con cláusula ON

- La condición de Join para un Natural Join es un equijoin con todas las tablas que tienen columnas iguales.
- Use la cláusula `ON` para especificar condiciones arbitrarias o especificar columnas de Join
- La condición de Join está separada de otras condiciones de búsqueda.
- La cláusula ON hace que el código sea fácil de entender

Recuperando registros con cláusula ON

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...

19 rows selected.

Self-Joins usando la cláusula ON

EMPLOYEES (WORKER)

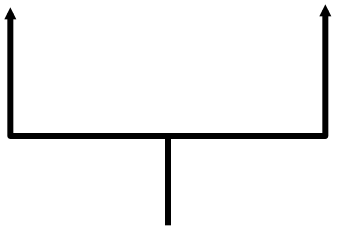
EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...



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

Self-Joins usando la cláusula ON

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON     (e.manager_id = m.employee_id);
```

EMP	MGR
Hartstein	King
Zlotkey	King
Mourgos	King
De Haan	King
Kochhar	King

...

19 rows selected.

Aplicando condiciones adicionales a un Join

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149 ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel	80	80	2500
176	Taylor	80	80	2500

Creando Joins de tres tablas con cláusula ON

```
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;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

...

19 rows selected.

Nonequijoins

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Salary in the EMPLOYEES table must be between lowest salary and highest salary in the JOB_GRADES table.

Recuperando registros con Nonequijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...

20 rows selected.

Outer Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

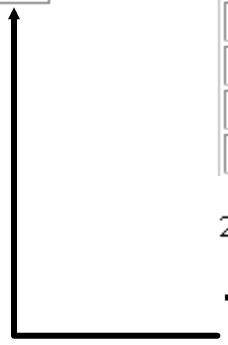
8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...

20 rows selected.



There are no employees in department 190.

INNER Versus OUTER Joins

- In SQL:1999, el join de dos tablas que recupera únicamente tuplas coincidentes es el Inner Join.
- Un join de dos tablas que recupera tuplas coincidentes y tuplas no coincidente con la tabla izquierda (o derecha) se le llama un left (o right) outer join.
- Un join entre dos tablas de un inner join junto con los resultados del left y right outer join es un full outer join.

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Davies	50	Shipping
...		
Kochhar	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
	190	Contracting

20 rows selected.

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
	190	Contracting

21 rows selected.

Productos Cartesianos

- Un producto cartesiano se forma:
  - La condición Join se omite
  - La condición Join es inválida
  - Cada una de las filas de la primera tabla son coincidentes con todas las filas de la segunda tabla
- Para evitar un producto cartesiano, la condición Join debe ser válida

Generando un Producto Cartesiano

EMPLOYEES (20 rows)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

20 rows selected.

DEPARTMENTS (8 rows)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

8 rows selected.

Cartesian product:
20 x 8 = 160 rows

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
100	90	1700
101	90	1700
102	90	1700
103	60	1700
104	60	1700
107	60	1700
...		

160 rows selected.

Creando Productos Cartesianos

- La cláusula `CROSS JOIN` produce el producto cartesiano de dos tablas

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

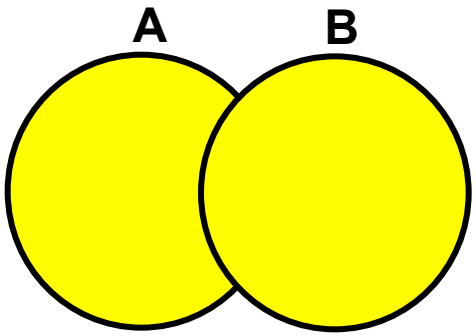
LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration

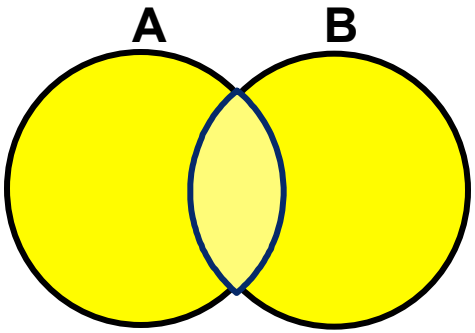
...

160 rows selected.

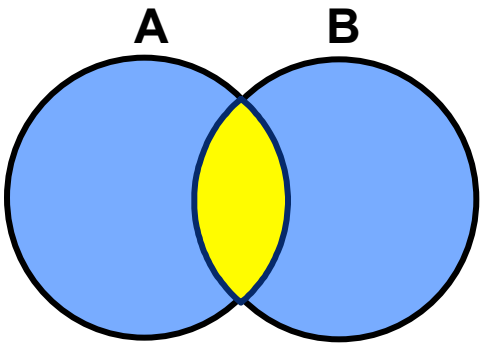
Operadores de Conjuntos

Operadores de Conjuntos

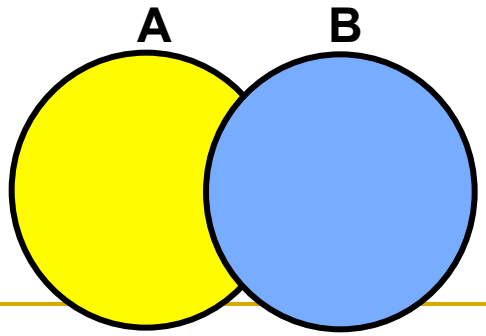




UNION/UNION ALL



INTERSECT

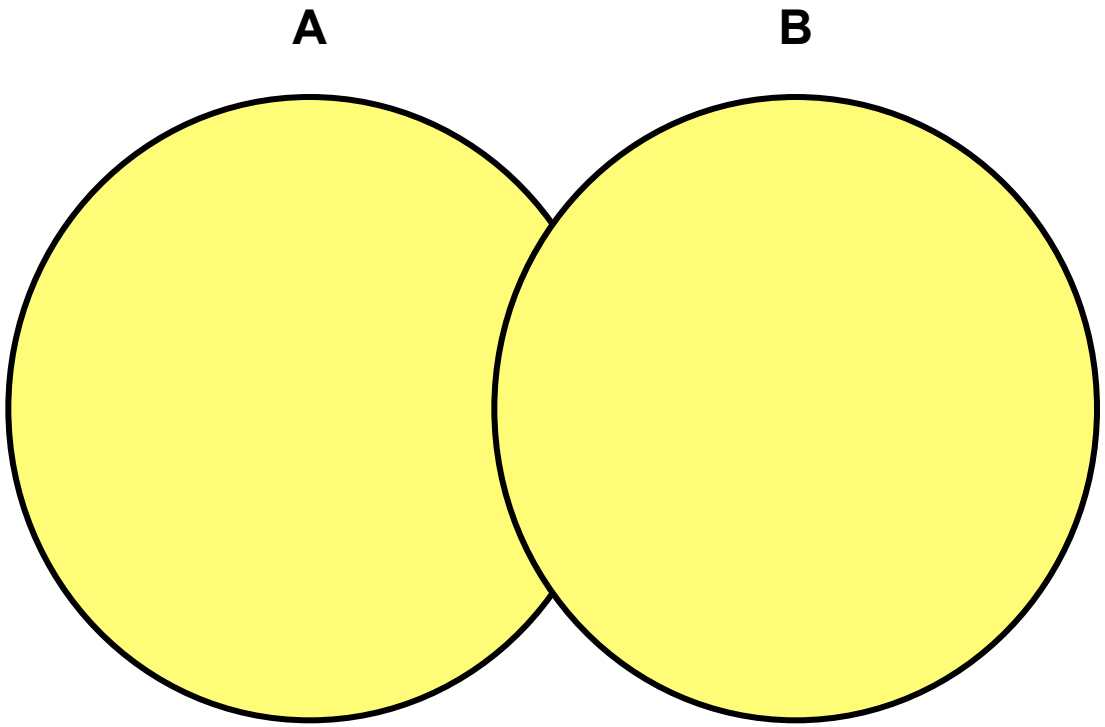


MINUS

Tablas utilizadas en el ejemplo

- Las tablas son:
  - `EMPLOYEES`: Detalles sobre empleados actuales.
  - `JOB_HISTORY`: Registra todos los detalles de fecha inicio y fecha fin del trabajo anterior, y el número de identificación del trabajo y del departamento cuando un empleado cambia de trabajo.

Operador UNION



El operador `UNION` retorna los resultados de las dos consultas luego de eliminar duplicados

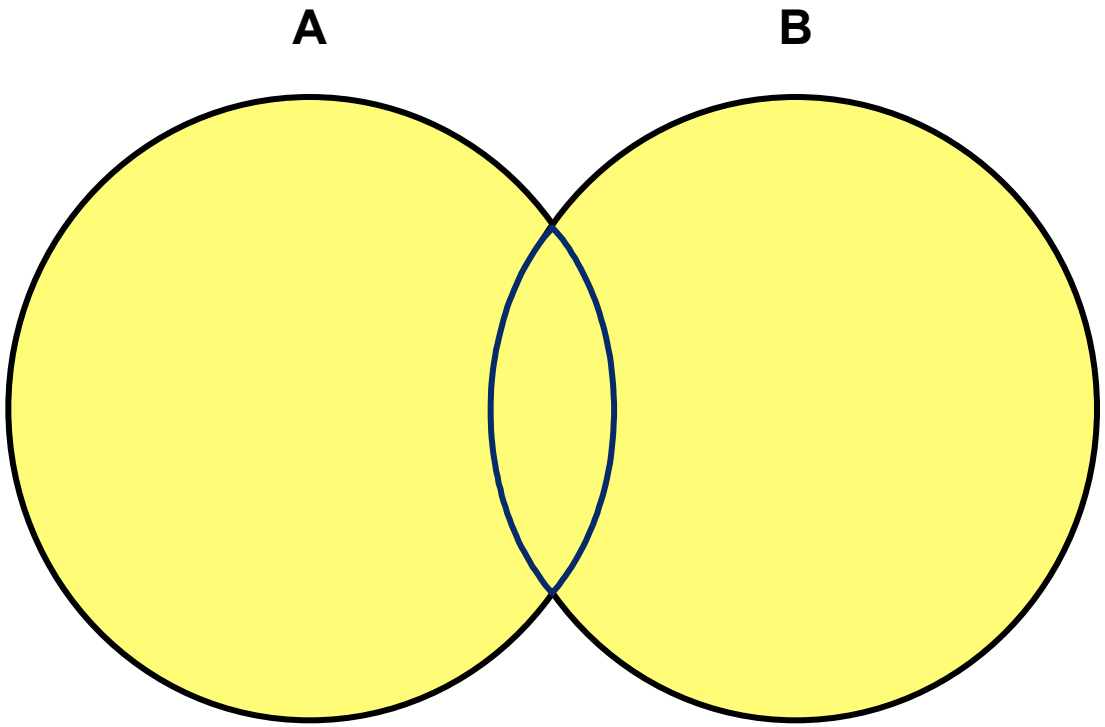
Usando el operador UNION

- Muestre todos los detalles del trabajo actual y el anterior del empleado.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AC_ACCOUNT
...	
200	AC_ACCOUNT
200	AD_ASST
...	
205	AC_MGR
206	AC_ACCOUNT

Operador UNION ALL



El operador `UNION ALL` operator retorna resultados de ambas consultas incluyendo los duplicados

Usando el operador UNION ALL

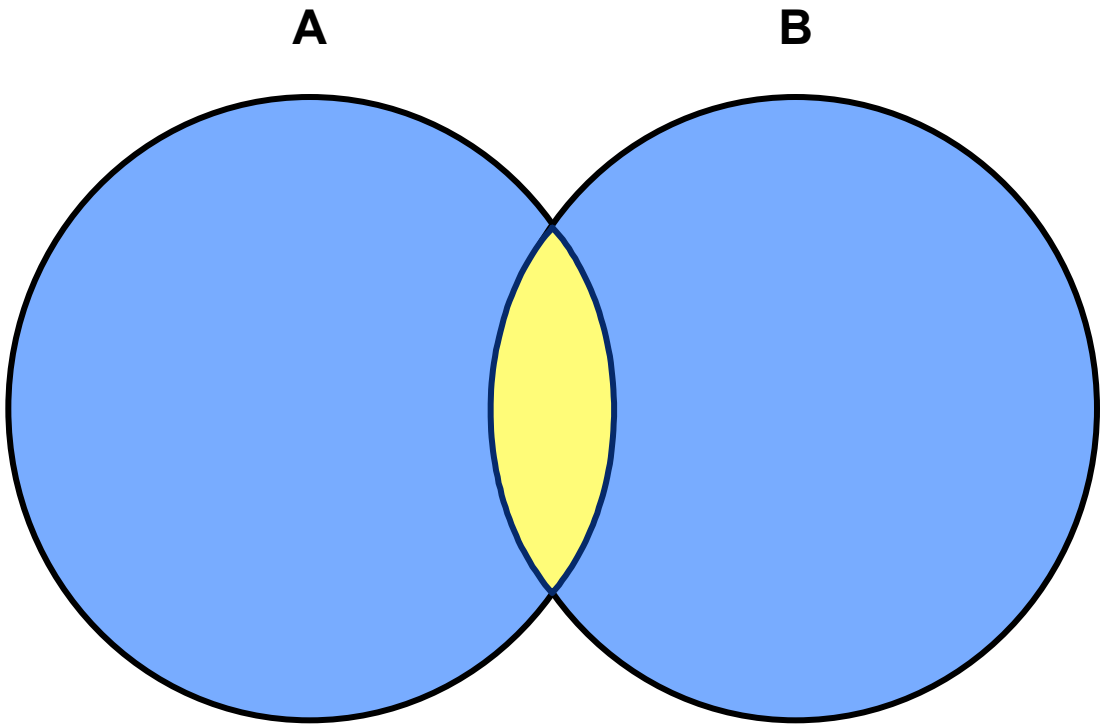
- Muestre el departamento actual y los anteriores de todos los empleados.

```
SELECT employee_id, job_id, department_id
FROM   employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM   job_history
ORDER BY  employee_id;
```

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
100	AD_PRES	90
101	AD_VP	90
...		
200	AD_ASST	10
200	AD_ASST	90
200	AC_ACCOUNT	90
...		
205	AC_MGR	110
206	AC_ACCOUNT	110

30 rows selected.

Operador INTERSECT



El operador `INTERSECT` retorna las tuplas comunes de ambas consultas.

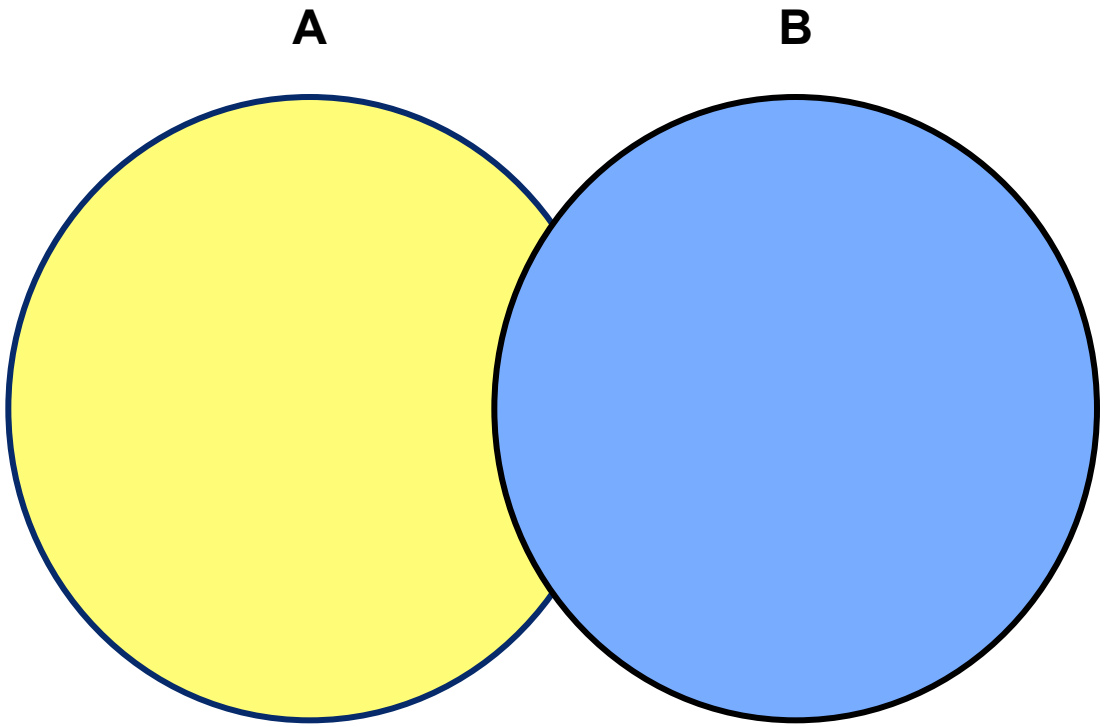
Usando el operador INTERSECT

- Muestre los IDs de empleados y los IDs de los trabajos de aquellos empleados que actualmente tienen un título de trabajo que es el mismo que el título de cuando los emplearon inicialmente (esto es, cambiaron trabajos pero ahora volvieron a su trabajo original).

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID
176	SA_REP
200	AD_ASST

Operador MINUS



El opoerador MINUS retorna tuplas d ela primera consulta que no están en la segunda

Operador MINUS

- Muestre los IDs de los empleados que no han cambiado de empleo nunca

```
SELECT employee_id
FROM   employees
MINUS
SELECT employee_id
FROM   job_history;
```

EMPLOYEE_ID
100
103
104
107
...
205
206

15 rows selected.

Lineamientos operadores de conjuntos

- Las expresiones en la lista de SELECT deben ser el mismo número y tipo.
- Se pueden usar paréntesis para alterar el orden de ejecución.
- La cláusula `ORDER BY`:
  - Puede aparecer solamente al final de la instrucción.
  - Acepta nombre de la columna, aliases del primer `SELECT` o notación posicional.

Matching los SELECT

- Usando el operador UNION despliegue los department_id, location_id y hire date de todos los empleados.

```
SELECT department_id, TO_NUMBER(null)
       location, hire_date
FROM   employees
UNION
SELECT department_id, location_id,  TO_DATE(null)
FROM   departments;
```

DEPARTMENT_ID	LOCATION	HIRE_DATE
10	1700	
10		17-SEP-87
20	1800	
20		17-FEB-96
...		
110	1700	
110		07-JUN-94
190	1700	
		24-MAY-99

27 rows selected.

Matching los SELECT Ejemplo

- Usando el operador UNION despliegue el employee ID, job ID, y salary de todos los empleados.

```
SELECT employee_id, job_id,salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID	SALARY
100	AD_PRES	24000
101	AC_ACCOUNT	0
101	AC_MGR	0
...		
205	AC_MGR	12000
206	AC_ACCOUNT	8300

30 rows selected.