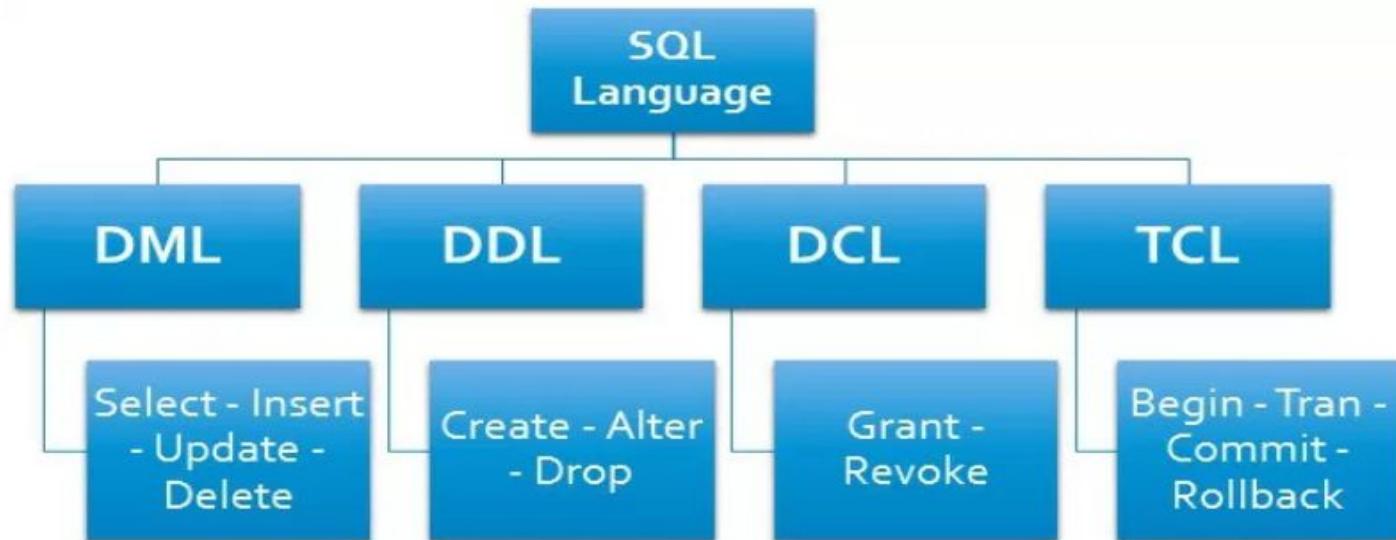


SQL

DDL - DML - DCL

Lenguajes en Bases de Datos Relacionales

SENTENCIAS SQL (DDL, DML, DCL Y TCL)



Sentencias

SELECT
INSERT
UPDATE
DELETE
MERGE

Data manipulation language (DML)

CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT

Data definition language (DDL)

GRANT
REVOKE

Data control language (DCL)

COMMIT
ROLLBACK
SAVEPOINT

Transaction control

Gestionando Objetos

DDL

Objetos en las Bases de Datos

Objeto	Descripción
Table	Unidad básica de almacenamiento; compuesto de filas
View	Representa lógicamente subconjuntos de datos de una o más tablas
Sequence	Genera valores numéricos.
Index	Mejora el rendimiento de algunas consultas.
Synonym	Da nombres alternativos a los objetos.

Creando tablas

```
CREATE TABLE table_name (  
  
    column1 datatype,  
  
    column2 datatype,  
  
    column3 datatype,  
  
    ....  
  
) ;
```

```
CREATE TABLE Customers (  
  
    personID int,  
  
    lastName varchar(255),  
  
    firstName varchar(255),  
  
    address varchar(255),  
  
    city varchar(255)  
) ;
```

Tipos de datos

SQL posee varios tipos de datos para almacenar información, los tipos de datos pueden ser:

- Numéricos (con o sin decimales).
- Alfanuméricos.
- Fecha y Hora
- Lógico

Además, la mayoría de gestores de BD actuales soportan el tipo: BLOB (Binary Large Object), para almacenar archivos.

Dependiendo de cada gestor de bases de datos en general se pueden tener los siguientes tipos de datos:

Númericos	Alfanúmericos	Fecha	Lógico	BLOB
Integer	char(n)	Date	Bit	Image
Numeric(n,m)	varchar(n)	DateTime		Text
Decimal(n)				
Float				

Modificando la tabla

```
ALTER TABLE table_name ADD column_name datatype;
```

```
ALTER TABLE Customers ADD Email varchar(255);
```

```
ALTER TABLE table_name MODIFY COLUMN column_name datatype;
```

```
ALTER TABLE Customers MODIFY COLUMN Email varchar(100);
```

```
ALTER TABLE table_name DROP COLUMN column_name;
```

```
ALTER TABLE Customers DROP COLUMN Email;
```

Borrando Tablas

```
DROP TABLE table_name;
```

```
ALTER TABLE Customers;
```

La sentencia TRUNCATE es usada para eliminar los datos dentro de la tabla, pero no para borrar la tabla en sí

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE TABLE Customers;
```

Restricciones en Tablas (Constraints)

Las restricciones de SQL se utilizan para especificar reglas para los datos en una tabla.

Las restricciones se utilizan para limitar el tipo de datos que pueden incluirse en una tabla. Esto asegura la precisión y confiabilidad de los datos en la tabla. Si hay alguna violación entre la restricción y la acción de datos, la acción se cancela.

Las restricciones pueden ser de nivel de columna o de tabla. Las restricciones de nivel de columna se aplican a una columna y las restricciones de nivel de tabla se aplican a toda la tabla.

Las siguientes restricciones se usan comúnmente en SQL:

- **NOT NULL**: garantiza que una columna no pueda tener un valor NULL
- **UNIQUE**: garantiza que todos los valores de una columna sean diferentes
- **PRIMARY KEY**: una combinación de NOT NULL y UNIQUE. Identifica de forma única cada fila en una tabla
- **FOREIGN KEY**: evita acciones que destruyan enlaces entre tablas
- **CHECK**: garantiza que los valores de una columna satisfagan una condición específica
- **DEFAULT**: establece un valor predeterminado para una columna si no se especifica ningún valor

Restricciones en Tablas (Constraints)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);

ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);

ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);

ALTER TABLE Persons
DROP CONSTRAINT UC_Person;
```

Restricciones en Tablas (Constraints)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID)
);

ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);

ALTER TABLE Persons DROP CONSTRAINT PK_Person;
```

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    CONSTRAINT PK_Orders PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
        REFERENCES Persons (ID)
);

ALTER TABLE Orders
ADD FOREIGN KEY (PersonID)
    REFERENCES Persons (ID);

ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

Restricciones en Tablas (Constraints)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18)
);

ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge
    CHECK (Age>=18 AND City='Sandnes');

ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);

ALTER TABLE Persons
MODIFY COLUMN City DEFAULT 'Sandnes';

ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

Vamos a ejercitarnos un poquito

- Crear una nueva base de datos
- Abrir la base de datos usando DBeaver CE
- Realizar cada una de las prácticas descritas.



Ejercicio 1

1. Crear la tabla **DEPT** con los campos descritos en la tabla anterior.
2. Aplique las restricciones NOT NULL y PRIMARY KEY a la tabla

Column Name	ID	NAME
Key Type	Primary key	
Nulls/Unique		
FK Table		
FK Column		
Data type	NUMBER	VARCHAR2
Length	7	25

Ejercicio 2

1. Crear la tabla **EMP** con los campos descritos en la tabla anterior.
2. Agregar la restricción de llave primaria a la tabla **EMP** en el campo ID
3. Agregar la restricción de llave foránea a la columna DEPT_ID al ID de la tabla **DEPT**.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				DEPT
FK Column				ID
Data type	NUMBER	VARCHAR2	VARCHAR2	NUMBER
Length	7	25	25	7

Ejercicio 3

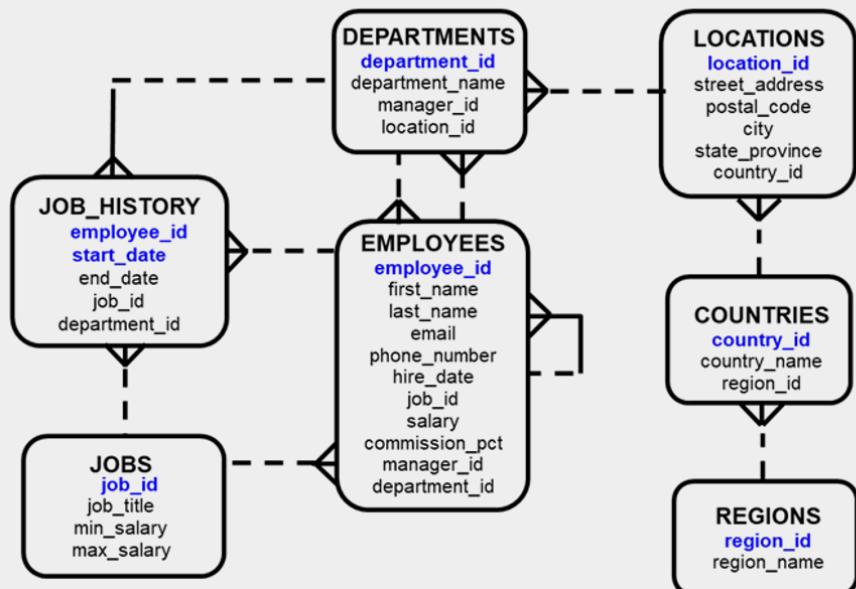
1. Intenta eliminar la tabla DEPT.
 - a. ¿Pudo realizarse el borrado?
 - b. ¿Cuál fue la razón de este comportamiento?
2. Elimina la tabla EMP.
 - a. ¿Pudo realizarse el borrado?
 - b. ¿Por qué ahora se logró borrar?
3. Elimina la tabla DEPT.

Realizando consultas
SELECT

Modelo de Datos

Estructura base de datos de
ejemplo

The Human Resources (HR) Schema



Selección

Tabla 1

Proyección

	C1	C2	C3	C4	C5	C6
R1	Light Green	Light Green	Pink	Pink	Light Green	Light Green
R2	Light Green	Light Green	Pink	Pink	Light Green	Light Green
R3	Light Green	Light Green	Pink	Pink	Light Green	Light Green
R4	Light Green	Light Green	Pink	Pink	Light Green	Light Green
R5	Light Green	Light Green	Pink	Pink	Light Green	Light Green
R6	Light Green	Light Green	Pink	Pink	Light Green	Light Green

Tabla 1

A 10x10 grid of colored squares. The first nine columns are light green, and the last column is magenta.

Tabla 1



A diagram showing a 10x10 grid of light blue squares. A thick vertical border of pink squares runs along the left edge of the grid. An arrow points from the left towards this pink border.

Tabla 2

Capacidades de las sentencias SELECT

Selección

```
SELECT *
FROM departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

Proyección

```
SELECT department_id, location_id  
FROM departments;
```

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

Expresiones Aritméticas

```
SELECT last_name, salary, salary + 300  
FROM employees;
```

	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900
...			

Operadores Aritméticos

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Precedencia de operadores aritméticos

- Multiplicación y División ocurren antes de Suma y Resta
- Operadores con la misma prioridad se evalúan de izquierda a derecha
- Los paréntesis son usados para modificar la precedencia o para aclarar la sentencia

```
SELECT last_name, salary, 12*salary+100  
FROM employees;
```

1

LAST_NAME	SALARY	12*SALARY+100
1 King	24000	288100
2 Kochhar	17000	204100
3 De Haan	17000	204100

...

```
SELECT last_name, salary, 12* (salary+100)  
FROM employees;
```

2

LAST_NAME	SALARY	12*(SALARY+100)
1 King	24000	289200
2 Kochhar	17000	205200
3 De Haan	17000	205200

...

Valores Nulos

```
SELECT last_name, job_id, salary, commission_pct  
FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)

...

12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2

...

19	Higgins	AC_MGR	12000	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

- NULL es un valor **no disponible, sin asignar, desconocido o no aplicable.**
- NULL **no significa** cero (0) ni espacio en blanco

```
SELECT last_name, 12*salary*commission_pct  
FROM employees;
```

LAST_NAME	12*SALARY*COMMISSION_PCT
1 King	(null)
2 Kochhar	(null)

...

12 Zlotkey	25200
13 Abel	39600
14 Taylor	20640

...

19 Higgins	(null)
20 Gietz	(null)

Expresiones aritméticas con valores Nulos siempre será NULL

Alias de Columnas

```
SELECT last_name AS name, commission_pct comm  
FROM employees;
```

	A Z NAME	A Z COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)

...

```
SELECT last_name "Name" , salary*12 "Annual Salary"  
FROM employees;
```

	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000

...

Operador de Concatenación

```
SELECT      last_name || job_id AS "Employees"  
FROM        employees;
```

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP

...

Literales de cadena

```
SELECT last_name || ' is a ' || job_id  
      AS "Employee Details"  
FROM   employees;
```

A Z	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP
...	

18	Vargas is a ST_CLERK
19	Whalen is a AD_ASST
20	Zlotkey is a SA_MAN

Eliminar resultados repetidos

```
SELECT department_id  
FROM employees;
```

1

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60

...

```
SELECT DISTINCT department_id  
FROM employees;
```

2

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110

...

Vamos a ejercitarnos otro poquito

- Conectarse a la base de datos HR en DBeaver.
- Realizar las consultas propuestas en el ejercicio



Ejercicio básicos

1. El departamento de recursos humanos desea que una consulta muestre el apellido, la identificación del trabajo, la fecha de contratación y la identificación del empleado para cada empleado, apareciendo primero la identificación del empleado. Proporcione un alias STARTDATE para la columna HIRE_DATE.
2. El departamento de recursos humanos desea que una consulta muestre todos los ID de trabajo únicos de la tabla EMPLOYEES.

Ejercicios avanzados

3. El departamento de recursos humanos quiere encabezados de columna más descriptivos para su informe sobre los empleados (consulta 1.1). Nombre los encabezados de columna Emp #, Employee, Job y Hire Date, respectivamente.
4. El departamento de recursos humanos ha solicitado un informe de todos los empleados y sus ID de trabajo. Muestre el apellido concatenado con la ID del trabajo (separados por una coma y un espacio) y nombre la columna Employee and Title.
5. Para familiarizarse con los datos en la tabla EMPLOYEES, cree una consulta para mostrar todos los datos de esa tabla. Separe cada columna de salida por una coma. Nombre el título de la columna THE_OUTPUT.

Restringiendo y
ordenando datos

Limitando filas usando una selección

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

...

“Obtener los
empleados en el
departamento 90”

```
SELECT employee_id, last_name, job_id, department_id  
FROM employees  
WHERE department_id = 90 ;
```



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

Cadenas de caracteres y fechas

```
SELECT last_name, job_id, department_id  
FROM   employees  
WHERE  last_name = 'Whalen' ;
```

```
SELECT last_name  
FROM   employees  
WHERE  hire_date = '17-FEB-96' ;
```

Formato de fecha por defecto: DD-MON-RR

Operadores de Comparación

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ... AND ...	Between two values (inclusive)
IN (set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Operadores de Comparación

```
SELECT last_name, salary  
FROM employees  
WHERE salary BETWEEN 2500 AND 3500 ;
```

```
SELECT last_name, salary  
FROM employees  
WHERE salary <= 3000 ;
```

Lower limit Upper limit

LAST_NAME	SALARY
1 Matos	2600
2 Vargas	2500

LAST_NAME	SALARY
1 Rajs	3500
2 Davies	3100
3 Matos	2600
4 Vargas	2500

Operadores de Comparación

```
SELECT employee_id, last_name, salary, manager_id  
FROM employees  
WHERE manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12000	101
8	202	Fay	6000	201

Operadores de Comparación (LIKE)

```
SELECT first_name  
FROM employees  
WHERE first_name LIKE 'S%' ;
```

```
SELECT last_name  
FROM employees  
WHERE last_name LIKE '_o%' ;
```

% representa cero o muchos caracteres

_ representa un carácter

LAST_NAME
Kochhar
Lorentz
Mourgos

Operadores de Comparación (IS NULL, IS NOT NULL)

```
1 SELECT last_name, manager_id
2 FROM employees
3 WHERE manager_id IS NULL
4
```

LAST_NAME	MANAGER_ID
King	-

[Download CSV](#)

```
1 SELECT last_name, manager_id
2 FROM employees
3 WHERE manager_id IS NOT NULL
4
```

LAST_NAME	MANAGER_ID
OConnell	124
Grant	124
Whalen	101
Hartstein	100
Fay	201
Mavris	101

Operadores Lógicos

Operator	Meaning
AND	Returns TRUE if <i>both</i> component conditions are true
OR	Returns TRUE if <i>either</i> component condition is true
NOT	Returns TRUE if the condition is false

Operadores Lógicos

```
SELECT employee_id, last_name, job_id, salary  
FROM employees  
WHERE salary >= 10000  
AND job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

Operadores Lógicos

```
SELECT employee_id, last_name, job_id, salary  
FROM employees  
WHERE salary >= 10000  
OR job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100 King	AD_PRES	24000	
2	101 Kochhar	AD_VP	17000	
3	102 De Haan	AD_VP	17000	
4	124 Mourgos	ST_MAN	5800	
5	149 Zlotkey	SA_MAN	10500	
6	174 Abel	SA_REP	11000	
7	201 Hartstein	MK_MAN	13000	
8	205 Higgins	AC_MGR	12000	

Operadores Lógicos

```
SELECT last_name, job_id  
FROM employees  
WHERE job_id  
      NOT IN ('IT_PROG', 'ST_CLERK', 'SA REP') ;
```

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

Reglas de Precedencia

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

Los paréntesis son usados para modificar la precedencia

Leyes de precedencia

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE job_id = 'SA_REP'  
OR job_id = 'AD_PRES'  
AND salary > 15000;
```

1

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000
2	Abel	SA_REP	11000
3	Taylor	SA_REP	8600
4	Grant	SA_REP	7000

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE (job_id = 'SA_REP'  
OR job_id = 'AD_PRES')  
AND salary > 15000;
```

2

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000

Ordenando filas

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

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	King	AD_PRES		90 17-JUN-87
2	Whalen	AD_ASST		10 17-SEP-87
3	Kochhar	AD_VP		90 21-SEP-89
4	Hunold	IT_PROG		60 03-JAN-90
5	Ernst	IT_PROG		60 21-MAY-91
6	De Haan	AD_VP		90 13-JAN-93

...

- ORDER BY es la última cláusula de la sentencia SELECT
- Las filas son ordenadas:
 - **ASC**: Orden ascendente (por defecto)
 - **DESC**: Orden descendente

Ordenando filas

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

1

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

2

```
SELECT      last_name, job_id, department_id, hire_date  
FROM        employees  
ORDER BY    3;
```

3

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

4

Vamos a ejercitarnos otro poquito

- Conectarse a la base de datos HR en DBeaver.
- Realizar las consultas propuestas en el ejercicio



Ejercicio básicos

1. Debido a problemas de presupuesto, el departamento de recursos humanos necesita un informe que muestre el apellido y el salario de los empleados que ganan más de \$ 12.000.
2. Cree un informe que muestre el apellido y el número de departamento para el número de empleado 176.
3. El departamento de recursos humanos necesita encontrar empleados con sueldos altos y bajos. Crea un informe para mostrar el apellido y el salario de cualquier empleado cuyo salario no esté en el rango de \$ 5.000 a \$ 12.000.

Ejercicio básicos

4. Cree un informe para mostrar el apellido, la identificación del trabajo y la fecha de contratación para los empleados con los apellidos de Matos y Taylor. Ordene la consulta en orden ascendente por la fecha de contratación.
5. Muestra el apellido y la identificación del departamento de todos los empleados en los departamentos 20 o 50 en orden alfabético ascendente por nombre.
6. Crea un informe para mostrar el apellido y el salario de los empleados que ganan entre \$ 5.000 y \$ 12.000, y están en el departamento 20 o 50. Etiquete las columnas Employee y Monthly Salary, respectivamente.

Ejercicio básicos

7. El departamento de recursos humanos necesita un informe que muestre el apellido y la fecha de contratación de todos los empleados que fueron contratados en 1994.
8. Cree un informe para mostrar el apellido y el cargo de todos los empleados que no tienen un gerente.
9. Cree un informe para mostrar el apellido, el salario y la comisión de todos los empleados que ganan comisiones. Ordenar datos en orden descendente de salario y comisiones.

Use la posición numérica de la columna en la cláusula ORDER BY.

Ejercicio avanzados

10. Muestra todos los apellidos de los empleados en los que la tercera letra del nombre es "a".
11. Muestre los apellidos de todos los empleados que tienen tanto una "a" como una "e" en su apellido.
12. Muestre el apellido, el trabajo y el salario de todos los empleados cuyos trabajos sean los de un representante de ventas o de un empleado de bolsa, y cuyos salarios no sean iguales a \$ 2.500, \$ 3.500 o \$ 7.000.
13. Cree un informe para mostrar el apellido, el salario y la comisión de todos los empleados cuya comisión es del 20%.

Funciones

Funciones SQL

Funciones que afectan 1 solo registro	
Funciones Matemáticas	Funciones que le permiten manipular datos numéricos de manera más eficaz.
Funciones de Cadena	Funciones que le permiten manipular datos de cadena de manera más eficaz.
Funciones de Fecha	Funciones que le permiten manipular datos de fecha y hora de manera más eficaz.
Funciones de Comparación	COALESCE, DECODE, y NULLIF.
Funciones que afectan a varios registros (grupos)	
Funciones de Agregación	AVG(), COUNT(), MIN(), MAX(), y SUM().

Funciones Matemáticas

Función	Descripción
<code>ABS (x)</code>	Devuelve el valor absoluto
<code>MOD (x, y)</code>	Devuelve el resto (módulo) de un número dividido por otro.
<code>ROUND (x, y)</code>	Redondea un número a una precisión específica
<code>CEIL (x) CEILING (x)</code>	Redondea un flotante al valor entero superior más cercano
<code>FLOOR (x)</code>	Redondea un flotante al valor entero inferior más cercano
<code>TRUNCATE (x, y) TRUNC (x, y)</code>	Se trunca a un número específico de posiciones decimales.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM employees
WHERE job_id = 'SA_REP';
```

	LAST_NAME	SALARY	MOD(SALARY,5000)
1	Abel	11000	1000
2	Taylor	8600	3600
3	Grant	7000	2000

Otras funciones:

- ACOS (x)
- ASIN (x)
- ATAN (x)
- COS (x)
- COT (x)
- EXP
- LN (x)
- LOG (x)
- LOG10 (x)
- LOG2 (x)
- PI
- RAND ()
- SIGN (x)
- SIN (x)
- SQRT (x)
- TAN (x)

Funciones de Cadena

Función	Descripción
CONCAT(str1,str2,...);	Devuelve el resultado de la concatenación de dos o más cadenas.
LOWER(str) / UPPER(str) LCASE(str) / UCASE(str)	Convierte todos los caracteres de una cadena a minúsculas / mayúsculas respectivamente.
LENGTH(str) LEN(str)	Devuelve el número de caracteres de una cadena determinada.
SUBSTRING(str,pos,length);	Extrae una subcadena de una cadena
REPLACE(str,old,new);	Reemplaza todas las apariciones de una subcadena especificada en una cadena por una nueva subcadena
LTRIM([LEADING TRAILING BOTH] char FROM str);	Elimina caracteres no deseados, por ejemplo, espacios en blanco de una cadena.

```
SELECT employee_id, last_name, department_id  
FROM employees  
WHERE last_name = 'higgins';  
0 rows selected
```

```
SELECT employee_id, last_name, department_id  
FROM employees  
WHERE LOWER(last_name) = 'higgins';
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

Otras funciones:

- ASCII(str) : Devuelve el código ASCII del primer carácter de una cadena.
- CHR(ascii) : Devuelve el carácter correspondiente al código ASCII de entrada.
- INSTR(str, search) : Devuelve la posición de la subcadena a buscar de la cadena origen.
- PRINTF(fmt, val1, ...) : Crea una cadena con el formato dado asignando los valores de cada plantilla %.

Funciones de Comparación

Función	Descripción
<code>COALESCE(arg1,arg2,...);</code>	Devuelve el primer argumento no nulo en la lista de argumentos.
<code>DECODE(e,s1,r1[,s2,r2],...,[,sn,rn] [,d]);</code>	Aprenda a agregar la lógica de procedimiento if-then-else a las consultas SQL.
<code>NULLIF(arg1,arg2);</code>	Compara dos expresiones y devuelve nulo si son iguales; de lo contrario, devuelve la primera expresión.
<code>IFNULL(arg1,arg2)</code> <code>NVL(arg1,arg2)</code>	Devuelve el valor de su primer argumento no NULL, o NULL si ambos argumentos son NULL.

```
SELECT article_id, title,  
       COALESCE(NULLIF(excerpt, ''),  
                  LEFT(body, 50)) AS summary  
FROM articles;
```

```
SELECT ID, product_name,  
       COALESCE( product_summary,  
                  LEFT (product_description, 50)) excerpt,  
       price, discount  
FROM products;
```

```
SELECT id, product_name, price, discount,  
       (price - COALESCE(discount, 0)) AS net_price  
FROM products;
```

```
SELECT employee_id, first_name, last_name, salary  
FROM employees  
ORDER BY DECODE('S',  
                  'F', first_name,  
                  'L', last_name,  
                  'S', salary);
```

Sentencia CASE

La instrucción **CASE** pasa por condiciones y devuelve un valor cuando se cumple la primera condición (como una instrucción **if-then-else**).

CASE

```
WHEN condition1 THEN result1
WHEN condition2 THEN result2
WHEN conditionN THEN resultN
ELSE result
END
```

Entonces, una vez que una condición es verdadera, dejará de leer y devolverá el resultado. Si no se cumple ninguna condición, devuelve el valor de la cláusula **ELSE**. Si no hay una parte **ELSE** y ninguna condición es verdadera, devuelve **NULL**.

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30
        THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY (CASE WHEN City IS NULL THEN Country ELSE City END);
```

Funciones anidadas

```
SELECT last_name,  
       UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))  
FROM   employees  
WHERE  department_id = 60;
```

	LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1	Hunold	HUNOLD_US
2	Ernst	ERNST_US
3	Lorentz	LORENTZ_US

Vamos a ejercitarnos otro poquito

- Conectarse a la base de datos HR en DBeaver.
- Realizar las consultas propuestas en el ejercicio



Ejercicio básicos

1. El departamento de recursos humanos necesita un informe para mostrar el número de empleado, apellido, salario y el salario aumentado en un 15.5% (expresado como un número entero) para cada empleado. Etiquete la columna **New Salary**.
2. Modifique la consulta anterior para agregar una columna que reste el salario anterior del nuevo salario. Nombre la columna **Increase**.
3. Escriba una consulta que muestre el apellido (con la primera letra en mayúscula y todas las demás letras en minúscula) y la longitud del apellido para todos los empleados cuyo nombre comienza con las letras "J", "A" o "M". Dé a cada columna un nombre apropiado. Ordene los resultados por los apellidos de los empleados.

Ejercicio básicos

4. El departamento de recursos humanos quiere encontrar la duración del empleo para cada empleado. Para cada empleado, muestre el apellido y calcule la cantidad de meses entre hoy y la fecha en que se contrató al empleado. Etiquete la columna como **MONTHS_WORKED**. Ordene sus resultados por la cantidad de meses empleados. Redondea el número de meses hasta el número entero más cercano.
5. Cree un informe que produzca lo siguiente para cada empleado: <apellido del empleado> gana <salario> mensualmente, pero quiere <3 veces el salario.>. Etiquete la columna **Dream Salaries**.
6. Muestre el apellido, la fecha de contratación y la fecha de revisión salarial de cada empleado, que es el primer lunes después de seis meses de servicio. Rotula la columna **REVIEW**. Formatee las fechas para que aparezcan en el formato similar a "Lunes, 31 de julio de 2000".

Ejercicio básicos

7. Muestre el apellido, la fecha de contratación y el día de la semana en que comenzó el empleado. Etiquete la columna **DAY**. Ordene los resultados por día de la semana, comenzando con el lunes.
8. Cree una consulta que muestre los apellidos y los montos de las comisiones de los empleados. Si un empleado no gana comisión, muestre "Sin comisión". Etiquete la columna **COMM**.

Ejercicio avanzados

9. Cree una consulta para mostrar el apellido y el salario de todos los empleados. Formatee el salario para que tenga 15 caracteres de largo, con el símbolo de \$ a la izquierda. Etiquete la columna **SALARY**. **Recomendación:** puede usar la función **PRINTF()**
10. Cree una consulta que muestre los primeros ocho caracteres de los apellidos de los empleados e indique los montos de sus salarios con asteriscos. Cada asterisco significa mil dólares. Ordene los datos en orden descendente de salario. Etiquete la columna **EMPLOYEES_AND_THEIR_SALARIES**.
11. Cree una consulta para mostrar el apellido y el número de semanas contratados para todos los empleados en el departamento 90. Etiquete el número de semanas en la columna **TENURE**. Trunca el valor de la cantidad de semanas a 0 decimales. Mostrar los registros en orden descendente de la tenencia del empleado.

Ejercicio avanzados

12. Escriba una consulta que muestre la calificación de todos los empleados en función del valor de la columna JOB_ID, utilizando los siguientes datos:

Job	Grade
Administration Vice President	A
Sales Manager	B
Programmer	C
Sales Representative	D
Stock Clerk	E
None of the above	0

Funciones Multi-fila (de grupo)

Funciones multi-fila (de grupo)

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	90	24000
2	90	17000
3	90	17000
4	60	9000
5	60	6000
6	60	4200
7	50	5800
8	50	3500
9	50	3100
10	50	2600
...		
18	20	6000
19	110	12000
20	110	8300

Maximum salary in
EMPLOYEES table

MAX(SALARY)
24000

Funciones multi-fila

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

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

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

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

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

Funciones multi-fila

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

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

A Z	COUNT(*)
1	5

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

A Z	COUNT(COMMISSION_PCT)
1	3

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

A Z	COUNT(DISTINCTDEPARTMENT_ID)
1	7

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

	DEPARTMENT_ID	Avg(SALARY)
1	(null)	7000
2	90	19333.3333333333...
3	20	9500
4	110	10150
5	50	3500
6	80	10033.3333333333...
7	60	6400
8	10	4400

GROUP BY

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

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

	AVG(SALARY)
1	7000
2	19333.33333333333333333333...
3	9500
4	10150
5	3500
6	10033.33333333333333333333...
7	6400
8	4400

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

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

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

```
SELECT department_id, job_id, COUNT(last_name)  
FROM employees  
GROUP BY department_id;
```

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

Errores comunes

```

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

```

GROUP BY HAVING

```

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

```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12000
4	80	11000

```

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
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

Vamos a ejercitarnos otro poquito

- Conectarse a la base de datos HR en DBeaver.
- Realizar las consultas propuestas en el ejercicio



Ejercicio básicos

1. Encuentre el salario más alto, más bajo, suma y promedio de todos los empleados. Etiquete las columnas **Maximum**, **Minimum**, **Sum** y **Average**, respectivamente. Redondea tus resultados al número entero más cercano.
2. Modifique la consulta anterior para mostrar el salario mínimo, máximo, suma y promedio para cada tipo de trabajo.
3. Escriba una consulta para mostrar el número de personas con el mismo trabajo.
4. Determine la cantidad de gerentes sin nombrarlos. Etiquete la columna **Number of Managers**.
Sugerencia: use la columna MANAGER_ID para determinar el número de administradores.
5. Encuentra la diferencia entre el salario más alto y el más bajo. Etiquete la columna **DIFFERENCE**.

Ejercicio avanzados

6. Cree un informe para mostrar el número de gerente y el salario del empleado con el salario más bajo para ese gerente. Excluir a cualquiera cuyo gerente no sea conocido. Excluya cualquier grupo donde el salario mínimo sea de \$ 6,000 o menos. Ordene la salida en orden descendente de salario.
7. Cree una consulta para mostrar el número total de empleados y, de ese total, el número de empleados contratados en 2005, 2006, 2007 y 2008. Cree encabezados de columna apropiados.
8. Cree una consulta matricial para mostrar el trabajo, el salario de ese trabajo según el número de departamento y el salario total de ese trabajo, para los departamentos 20, 50, 80 y 90, dando a cada columna un encabezado apropiado.

Job	Dept 20	Dept 50	Dept 80	Dept 90	Total
-----	---------	---------	---------	---------	-------

Mostrando datos
de múltiples
tablas

Uniendo tablas SQL

Natural joins

- NATURAL JOIN
- USING
- ON

Self-join

Nonequijoins

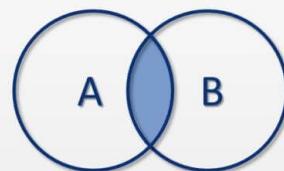
Outer join

- LEFT OUTER
- RIGHT OUTER
- FULL OUTER

Cross join

```
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];
```

INNER JOIN

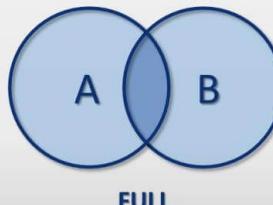


OUTER JOIN



LEFT

RIGHT



FULL

Natural Join

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

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

JOIN - USING

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

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	124	Mourgos	1500	50
5	144	Vargas	1500	50
6	143	Matos	1500	50
7	142	Davies	1500	50
8	141	Rajs	1500	50
9	107	Lorentz	1400	60
10	104	Ernst	1400	60
...				
19	205	Higgins	1700	110

```
SELECT l.city, d.department_name  
FROM   locations l JOIN departments d  
USING (location_id)  
WHERE d.location_id = 1400;
```



INNER JOIN / JOIN - 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_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	
10	104	Ernst	60	60	
...					

```
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;
```

SELF JOIN

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

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King
9	Hartstein	King
10	De Haan	King

...

NOEQUIJOIN

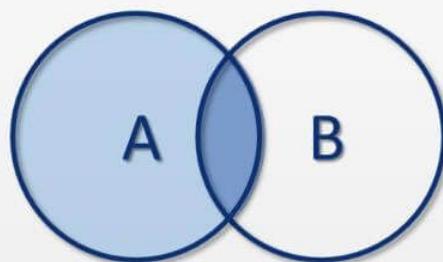
```
SELECT e.last_name, e.salary, j.grade_level  
FROM employees e JOIN job_grades j  
ON e.salary  
BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

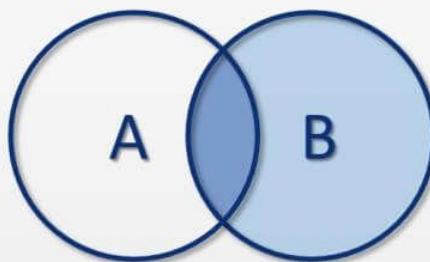
...

OUTER JOIN

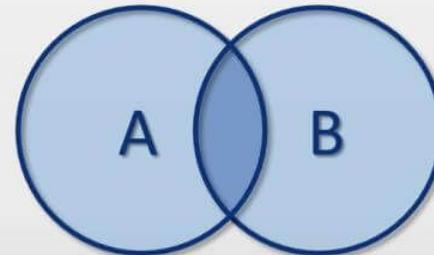
OUTER JOIN



LEFT



RIGHT



FULL

LEFT OUTER JOIN

```
1 SELECT e.last_name, e.department_id, d.department_name  
2 FROM employees e  
3 LEFT OUTER JOIN departments d ON (e.department_id = d.department_id);  
4
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Raphaely	30	Purchasing
Khoo	30	Purchasing
Baida	30	Purchasing
Tobias	30	Purchasing
■ ■ ■		

Popp	100	Finance
Higgins	110	Accounting
Gietz	110	Accounting
Grant	-	-

[Download CSV](#)

107 rows selected.

RIGHT OUTER JOIN

```
1 SELECT e.last_name, e.department_id, d.department_name  
2 FROM employees e  
3 RIGHT OUTER JOIN departments d ON (e.department_id = d.department_id);  
4
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
De Haan	90	Executive
Hunold	60	IT
Ernst	60	IT
Austin	60	IT
Pataballa	60	IT
Lorentz	60	IT
Greenberg	100	Finance
■ ■ ■		

Gietz	110	Accounting
-	-	IT Support
-	-	Operations
-	-	Payroll

FULL OUTER JOIN

```
1 SELECT e.last_name, e.department_id, d.department_name  
2 FROM employees e  
3 FULL OUTER JOIN departments d ON (e.department_id = d.department_id);  
4
```

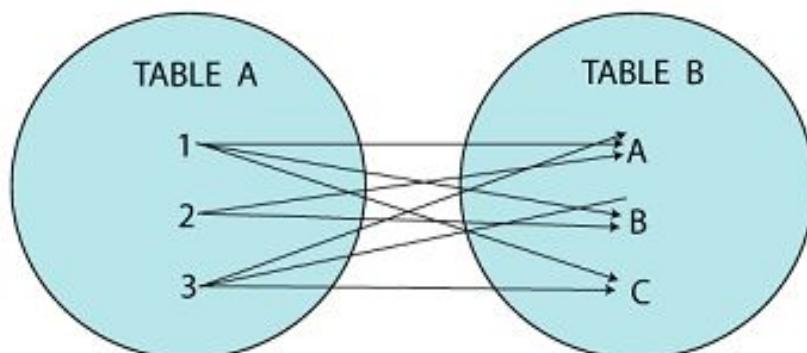
LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
De Haan	90	Executive
Hunold	60	IT
Ernst	60	IT
Austin	60	IT
Pataballa	60	IT
Lorentz	60	IT

■ ■ ■

Higgins	110	Accounting
Grant	-	-
-	-	Treasury
-	-	Manufacturing
-	-	Corporate Tax

CROSS JOIN

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



LAST_NAME	DEPARTMENT_NAME
1 Abel	Administration
2 Davies	Administration
3 De Haan	Administration
4 Ernst	Administration
5 Fay	Administration
...	

159	Whalen	Contracting
160	Zlotkey	Contracting

Vamos a ejercitarnos otro poquito

- Conectarse a la base de datos HR en DBeaver.
- Realizar las consultas propuestas en el ejercicio



Ejercicio básicos

1. Escriba una consulta para el departamento de recursos humanos para producir las direcciones de todos los departamentos. Utilice las tablas de LOCATIONS y COUNTRIES. Muestre la identificación de ubicación, la dirección, la ciudad, el estado o la provincia y el país en la salida. Use una NATURAL JOIN para producir los resultados.
2. El departamento de recursos humanos necesita un informe de sólo aquellos empleados con los departamentos correspondientes. Escriba una consulta para mostrar el apellido, el número de departamento y el nombre de departamento para estos empleados.
3. El departamento de recursos humanos necesita un informe de los empleados en Toronto. Muestre el apellido, el trabajo, el número de departamento y el nombre del departamento para todos los empleados que trabajan en Toronto.

Ejercicio básicos

4. Cree un informe para mostrar el apellido y el número del empleado junto con el apellido y el número del gerente. Etiquete las ***Employee***, ***Emp#***, ***Manager*** y ***Mgr#***, respectivamente.
5. Modifique el informe anterior para mostrar a todos los empleados, incluido King, que no tiene gerente. Ordene los resultados por el número de empleado.
6. Cree un informe para el departamento de recursos humanos que muestre los apellidos de los empleados, los números de departamento y todos los empleados que trabajan en el mismo departamento que un empleado determinado. Dé a cada columna una etiqueta apropiada.
7. El departamento de recursos humanos necesita un informe sobre las calificaciones laborales (tabla **JOB_GRADES**) y los salarios. Cree una consulta que muestre el nombre, el trabajo, el nombre del departamento, el salario y la calificación de todos los empleados.

Ejercicio avanzados

8. El departamento de recursos humanos quiere determinar los nombres de todos los empleados que fueron contratados después de Davies. Cree una consulta para mostrar el nombre y la fecha de contratación de cualquier empleado contratado después del empleado Davies.
9. El departamento de recursos humanos necesita encontrar los nombres y las fechas de contratación de todos los empleados que fueron contratados antes que sus gerentes, junto con los nombres y fechas de contratación de sus gerentes.

Subconsultas

Subconsultas

```
SELECT      select_list
FROM        table
WHERE       expr operator
           (SELECT      select_list
            FROM       table);
```

```
1 select salary
2 from employees
3 where last_name = 'Abel'
4
```

SALARY
11000

[Download CSV](#)

```
1 SELECT last_name, salary
2 FROM employees
3 WHERE salary > (
4 select salary
5 from employees
6 where last_name = 'Abel'
7 );
```

LAST_NAME	SALARY
King	24000

Subconsultas de fila única

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

```
1 SELECT last_name, job_id, salary
2 FROM employees
3 WHERE job_id = (
4     select job_id
5         from employees
6         where first_name = 'Jonathon'
7 )
8 AND salary > (
9     select salary
10        from employees
11       where first_name = 'Jonathon'
12 );
```

LAST_NAME	JOB_ID	SALARY
Tucker	SA_REP	10000
Bernstein	SA_REP	9500
Hall	SA_REP	9000
...	...	-----

```
1 SELECT last_name, job_id, salary
2 FROM employees
3 WHERE salary = (
4     SELECT MIN(salary)
5         FROM employees
6 );
```

LAST_NAME	JOB_ID	SALARY
Olson	ST_CLERK	2100

[Download CSV](#)

Subconsultas de fila única

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

```
1 SELECT department_id, MIN(salary)
2 FROM employees
3 GROUP BY department_id
4 HAVING MIN(salary) > (
5   SELECT MIN(salary)
6   FROM employees
7   WHERE department_id = 50
8 );|
```

DEPARTMENT_ID	MIN(SALARY)
40	6500
110	8300
90	17000

```
1 SELECT employee_id, last_name
2 FROM employees
3 WHERE salary = (
4   SELECT MIN(salary)
5   FROM employees
6   GROUP BY department_id
7 );|
```

ORA-01427: single-row subquery returns more than one row

Subconsultas multi-fila

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Compares a value to each value in a list or returned by a query. Evaluates to FALSE if the query returns no rows.
ALL	Must be preceded by =, !=, >, <, <=, >=. Compares a value to every value in a list or returned by a query. Evaluates to TRUE if the query returns no rows.

```
1 SELECT employee_id, first_name, last_name
2 FROM employees
3 WHERE department_id IN (1 , 3, 8, 10, 11)
4 ORDER BY first_name , last_name;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
200	Jennifer	Whalen

[Download CSV](#)

```
1 SELECT employee_id, first_name, last_name
2 FROM employees
3 WHERE department_id IN (
4   SELECT department_id
5   FROM departments
6   WHERE location_id = 1700
7 )
8 ORDER BY first_name , last_name;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
115	Alexander	Khoo
109	Daniel	Faviet

Subconsultas multi-fila

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Compares a value to each value in a list or returned by a query. Evaluates to FALSE if the query returns no rows.
ALL	Must be preceded by =, !=, >, <, <=, >=. Compares a value to every value in a list or returned by a query. Evaluates to TRUE if the query returns no rows.

```
1 SELECT employee_id, first_name, last_name
2 FROM employees
3 WHERE department_id NOT IN (
4     SELECT department_id
5     FROM departments
6     WHERE location_id = 1700
7 )
8 ORDER BY first_name , last_name;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
121	Adam	Fripp
102	Lex	De Haan

```
1 SELECT employee_id, first_name, last_name, salary
2 FROM employees
3 WHERE salary >= ALL (
4     SELECT MIN(salary)
5     FROM employees
6     GROUP BY department_id
7 )
8 ORDER BY first_name , last_name;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
102	Lex	De Haan	17000

Subconsultas multi-fila

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Compares a value to each value in a list or returned by a query. Evaluates to FALSE if the query returns no rows.
ALL	Must be preceded by =, !=, >, <, <=, >=. Compares a value to every value in a list or returned by a query. Evaluates to TRUE if the query returns no rows.

```
1 SELECT employee_id, first_name, last_name, salary
2 FROM employees
3 WHERE salary >= ANY(
4     SELECT MAX(salary)
5     FROM employees
6     GROUP BY department_id
7 );
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000

```
1 SELECT employee_id, first_name, last_name, salary
2 FROM employees
3 WHERE salary >= SOME(
4     SELECT MAX(salary)
5     FROM employees
6     GROUP BY department_id
7 );
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000

Subconsultas como tabla

```
1 SELECT ROUND(AVG(average_salary), 0)
2 FROM (
3     SELECT AVG(salary) average_salary
4     FROM employees
5     GROUP BY department_id
6 ) department_salary;
```

ROUND(AVG(AVERAGE_SALARY),0)
8153

[Download CSV](#)

Subconsultas como atributo

```
1  SELECT employee_id, first_name, last_name, salary,  
2      (SELECT ROUND(AVG(salary), 0) FROM employees) average_salary,  
3      salary - (SELECT ROUND(AVG(salary), 0) FROM employees) difference  
4  FROM employees  
5  ORDER BY first_name , last_name;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	AVERAGE_SALARY	DIFFERENCE
121	Adam	Fripp	8200	6462	1738

Subconsulta con comparación multicolumna

```
1  SELECT employee_id, manager_id, department_id
2  FROM employees
3  WHERE (manager_id, department_id) IN (
4      SELECT manager_id, department_id
5      FROM employees
6      WHERE first_name = 'John')
7  AND first_name <> 'John';
8
```

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
109	108	100
111	108	100
112	108	100

Subconsultas correlacionadas

```
1 SELECT employee_id, first_name, last_name  
2 FROM employees  
3 WHERE department_id IN (  
4     SELECT department_id  
5     FROM departments  
6     WHERE location_id = 1700  
7 )  
8 ORDER BY first_name , last_name;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
115	Alexander	Khoo
109	Daniel	Faviet

```
1 SELECT department_name  
2 FROM departments d  
3 WHERE NOT EXISTS(  
4     SELECT 1  
5     FROM employees e  
6     WHERE salary > 10000  
7     AND e.department_id = d.department_id  
8 )  
9 ORDER BY department_name;
```

DEPARTMENT_NAME
Administration
Benefits

Subconsultas correlacionadas

```
1 SELECT employee_id, first_name, last_name, department_name, salary,  
2   (SELECT ROUND(AVG(salary),0)  
3    FROM employees  
4    WHERE department_id = e.department_id) avg_salary_in_department  
5   FROM employees e  
6  INNER JOIN departments d ON d.department_id = e.department_id  
7 ORDER BY department_name, first_name, last_name;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_NAME	SALARY	AVG_SALARY_IN_DEPARTMENT
205	Shelley	Higgins	Accounting	12008	10154
206	William	Gietz	Accounting	8300	10154

```
1 select last_name  
2   from employees  
3  where department_id in (  
4    select department_id  
5      from departments  
6     where location_id in (  
7       select location_id  
8          from locations  
9        where country_id = (  
10       select country_id  
11          from countries  
12         where country_name='United Kingdom'  
13     )  
14   )  
15 );
```

LAST_NAME

Cláusula WITH

Usando la cláusula WITH, puedes reutilizar un bloque SELECT más de una vez en la misma consulta.

Los resultados obtenidos, son guardados temporalmente en memoria.

Se puede utilizar para mejorar el rendimiento.

```
1 WITH
2 dept_costs AS (
3     SELECT d.department_name, SUM(e.salary) AS dept_total
4     FROM employees e
5     JOIN departments d ON e.department_id = d.department_id
6     GROUP BY d.department_name
7 ),
8 avg_cost AS (
9     SELECT SUM(dept_total)/COUNT(*) AS dept_avg
10    FROM dept_costs
11 )
12 SELECT *
13 FROM dept_costs
14 WHERE dept_total > (
15     SELECT dept_avg
16     FROM avg_cost
17 )
18 ORDER BY department_name;
```

DEPARTMENT_NAME	DEPT_TOTAL
Sales	304500
Shipping	156400

[Download CSV](#)

2 rows selected.

Vamos a ejercitarnos otro poquito

- Conectarse a la base de datos HR en DBeaver.
- Realizar las consultas propuestas en el ejercicio



Ejercicio básicos

1. El departamento de recursos humanos necesita una consulta que solicite al usuario el apellido de un empleado. La consulta luego muestra el apellido y la fecha de contratación de cualquier empleado en el mismo departamento que el empleado cuyo nombre proporcionan (excluyendo a ese empleado). Por ejemplo, si el usuario ingresa a Zlotkey, busque todos los empleados que trabajan con Zlotkey (excluyendo Zlotkey).
2. Cree un informe que muestre el número de empleado, el apellido y el salario de todos los empleados que ganan más que el salario promedio. Ordene los resultados en orden de salario ascendente.
3. Escriba una consulta que muestre el número de empleado y el apellido de todos los empleados que trabajan en un departamento con cualquier empleado cuyo apellido contenga la letra "u".

Ejercicio básicos

4. El departamento de recursos humanos necesita un informe que muestre el apellido, el número de departamento y la identificación de trabajo de todos los empleados cuya identificación de ubicación de departamento es 1700.
5. Cree un informe para Recursos Humanos que muestre el apellido y el salario de cada empleado que se reporta a King.
6. Cree un informe para RR. HH. Que muestre el número de departamento, el apellido y la identificación del trabajo para cada empleado del departamento Ejecutivo.
7. Cree un informe que muestre una lista de todos los empleados cuyo salario es mayor que el de cualquier empleado del departamento 60.

Operaciones de conjuntos

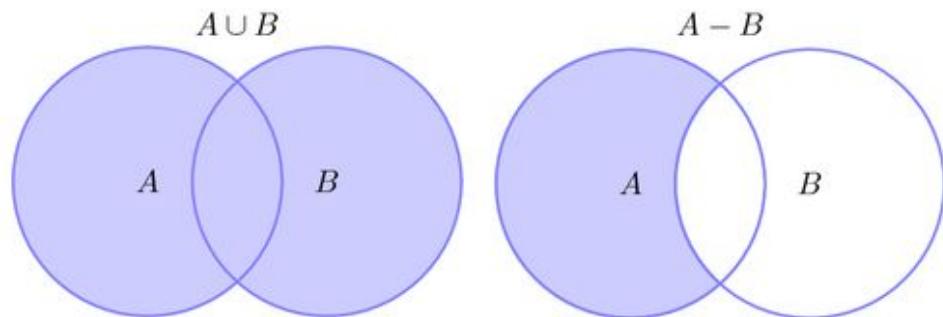
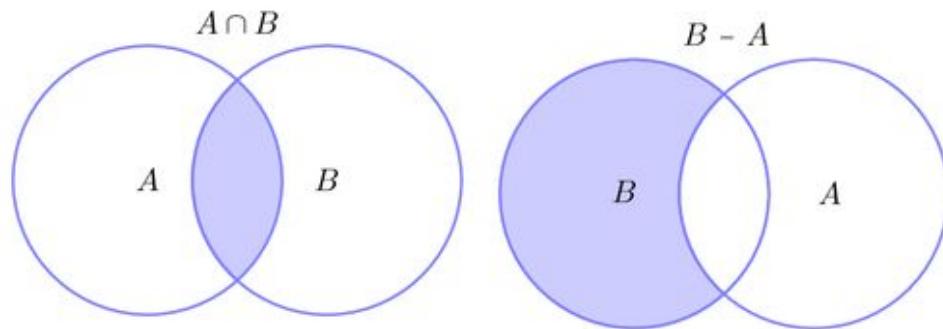
Operaciones de conjunto

UNION

UNION ALL

INTERSECT

MINUS



UNION

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

	EMPLOYEE_ID	JOB_ID
1		100 AD_PRES
2		101 AC_ACCOUNT
...		
22		200 AC_ACCOUNT
23		200 AD_ASST
24		201 MK_MAN
...		

UNION ALL

```
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
1	100	AD_PRES	90
...			
16	144	ST_CLERK	50
17	149	SA_MAN	80
18	174	SA_REP	80
19	176	SA_REP	80
20	176	SA_MAN	80
21	176	SA_REP	80
22	178	SA_REP	(null)
...			
30	206	AC_ACCOUNT	110

INTERSECT

```
SELECT employee_id, job_id  
FROM employees
```

INTERSECT

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

	EMPLOYEE_ID	JOB_ID
1		176 SA_REP
2		200 AD_ASST

MINUS

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

EMPLOYEE_ID
1
2
3
4
5
...

14	205
15	206

Características a tener en cuenta con operaciones de conjuntos

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

Columnas debe coincidir
en sus tipos

Si existe ordenamiento,
sólo debe existir una vez
y al final de la sentencia

```
SELECT location_id, department_name "Department",  
      TO_CHAR(NULL) "Warehouse location"  
FROM departments  
UNION  
SELECT location_id, TO_CHAR(NULL) "Department",  
      state_province  
FROM locations  
ORDER BY location_id, "Warehouse location";
```

Vamos a ejercitarnos otro poquito

- Conectarse a la base de datos HR en DBeaver.
- Realizar las consultas propuestas en el ejercicio



Ejercicio básicos

1. El departamento de recursos humanos necesita una lista de ID de departamento para los departamentos que no contienen la ID de trabajo '**Stock Clerk**'.
2. El departamento de recursos humanos necesita una lista de países que no tienen departamentos ubicados en ellos. Muestra la identificación del país y el nombre de los países.
3. Produzca una lista de trabajos para los departamentos 10, 50 y 20, en ese orden. Mostrar la ID del trabajo y la ID del departamento.

Ejercicio básicos

4. Cree un informe que enumere las identificaciones de los empleados y las identificaciones de los empleados que actualmente tienen un título de trabajo que es el mismo que el de su trabajo cuando fueron contratados inicialmente por la empresa (es decir, cambiaron de trabajo, pero ahora han vuelto a hacer su trabajo original).
5. El departamento de recursos humanos necesita un informe con las siguientes especificaciones:
 - a. Apellido e identificación del departamento de todos los empleados de la tabla EMPLOYEES, independientemente de si pertenecen o no a un departamento
 - b. ID de departamento y nombre de departamento de todos los departamentos de la tabla DEPARTMENTS, independientemente de si tienen empleados trabajando en ellos

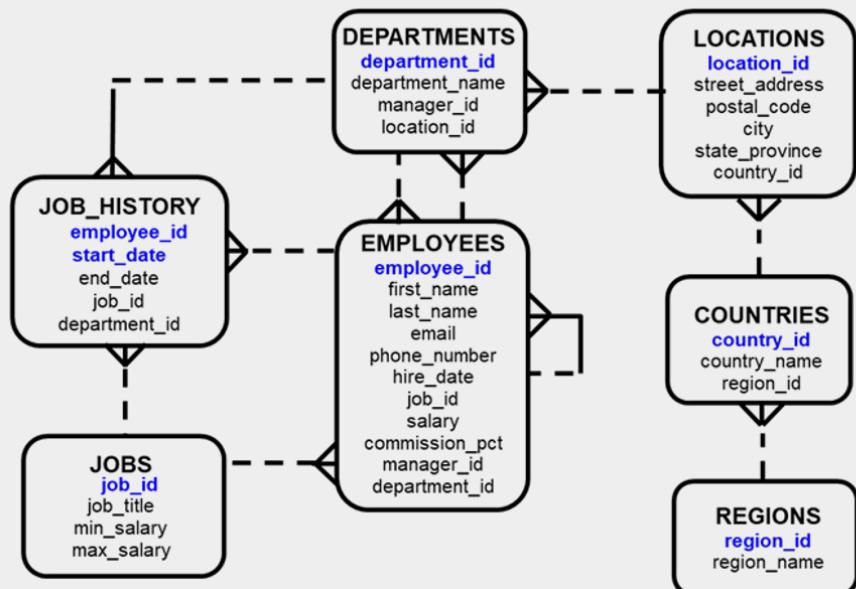
Escriba una consulta compuesta para lograr esto.

Manipulando datos DML

Modelo de Datos

Estructura base de datos de
ejemplo

The Human Resources (HR) Schema



Insertando nuevas filas

```
INSERT INTO table [(column [, column...])]  
VALUES (value [, value...]);
```

```
INSERT INTO departments(department_id,  
                      department_name, manager_id, location_id)  
VALUES (70, 'Public Relations', 100, 1700);  
1 rows inserted
```

```
INSERT INTO departments (department_id,  
                      department_name)  
VALUES (30, 'Purchasing');
```

```
INSERT INTO departments  
VALUES (100, 'Finance', NULL, NULL);  
1 rows inserted
```

Insertando nuevas filas

```
INSERT INTO table [(column [, column...])]  
VALUES  
      (value [, value...]);
```

```
INSERT INTO sales_reps(id, name, salary, commission_pct)  
SELECT employee_id, last_name, salary, commission_pct  
FROM employees  
WHERE job_id LIKE '%REP%';
```

```
4 rows inserted
```

Actualizando filas

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

```
UPDATE employees
SET     department_id = 50
WHERE   employee_id = 113;
```

```
1 rows updated
```

```
UPDATE      copy_emp
SET        department_id = 110;
```

```
22 rows updated
```

Actualizando filas

```
UPDATE      table  
SET         column = value [, column = value, ...]  
[WHERE      condition];
```

```
UPDATE      employees  
SET         job_id   = (SELECT    job_id  
                      FROM      employees  
                      WHERE     employee_id = 205),  
                  salary   = (SELECT    salary  
                      FROM      employees  
                      WHERE     employee_id = 205)  
WHERE      employee_id    = 113;
```

```
1 rows updated
```

Actualizando filas

```
UPDATE      table  
SET         column = value [, column = value, ...]  
[WHERE      condition];
```

```
UPDATE      copy_emp  
SET         department_id  =  (SELECT department_id  
                               FROM employees  
                               WHERE employee_id = 100)  
WHERE      job_id          =  (SELECT job_id  
                               FROM employees  
                               WHERE employee_id = 200);
```

```
1 rows updated
```

Eliminando filas

```
DELETE [FROM] table  
[WHERE] condition;
```

```
DELETE FROM departments  
WHERE department_name = 'Finance';
```

```
1 rows deleted
```

```
DELETE FROM copy_emp;
```

```
22 rows deleted
```

```
DELETE FROM employees  
WHERE department_id =  
      (SELECT department_id  
       FROM departments  
       WHERE department_name  
             LIKE '%Public%');
```

```
1 rows deleted
```

Vamos a ejercitarnos otro poquito

- Conectarse a la base de datos HR en DBeaver.
- Realizar las consultas propuestas en el ejercicio



Ejercicio básicos

1. Cree una instrucción INSERT para agregar la primera fila de datos a la tabla MY_EMPLOYEE a partir de los siguientes datos de muestra. No nombre las columnas en la cláusula INSERT. No ingrese todas las filas todavía.
2. Rellene la tabla MY_EMPLOYEE con la segunda fila de los datos de muestra de la lista anterior. Esta vez, enumere las columnas explícitamente en la cláusula INSERT.
3. Verifique su inserción a la tabla.

```
CREATE TABLE my_employee (
    id NUMBER(4) NOT NULL,
    last_name VARCHAR2(25),
    first_name VARCHAR2(25),
    userid VARCHAR2(8),
    salary NUMBER(9,2)
);
```

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

Ejercicio básicos

4. Rellene la tabla MY_EMPLOYEE con las siguientes dos filas de los datos de muestra de la lista anterior.
5. Verifique su inserción a la tabla.
6. Hacer que las adiciones de datos sean permanentes
7. Cambie el apellido del empleado 3 a Drexler.
8. Cambie el salario a \$ 1.000 para todos los empleados que tengan un salario inferior a \$ 900.
9. Verifique sus cambios en la tabla.

```
CREATE TABLE my_employee (
    id NUMBER(4) NOT NULL,
    last_name VARCHAR2(25),
    first_name VARCHAR2(25),
    userid VARCHAR2(8),
    salary NUMBER(9,2)
);
```

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

Ejercicio básicos

10. Eliminar Betty Dancs de la tabla MY_EMPLOYEE.
11. Verifique sus cambios en la tabla.
12. Rellene la tabla con la última fila de los datos de muestra.
13. Verifique su inserción a la tabla.
14. Elimine todas las filas de la tabla MY_EMPLOYEE.
15. Verifique que la tabla está vacía.

```
CREATE TABLE my_employee (
    id NUMBER(4) NOT NULL,
    last_name VARCHAR2(25),
    first_name VARCHAR2(25),
    userid VARCHAR2(8),
    salary NUMBER(9,2)
);
```

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

Transacciones TCL

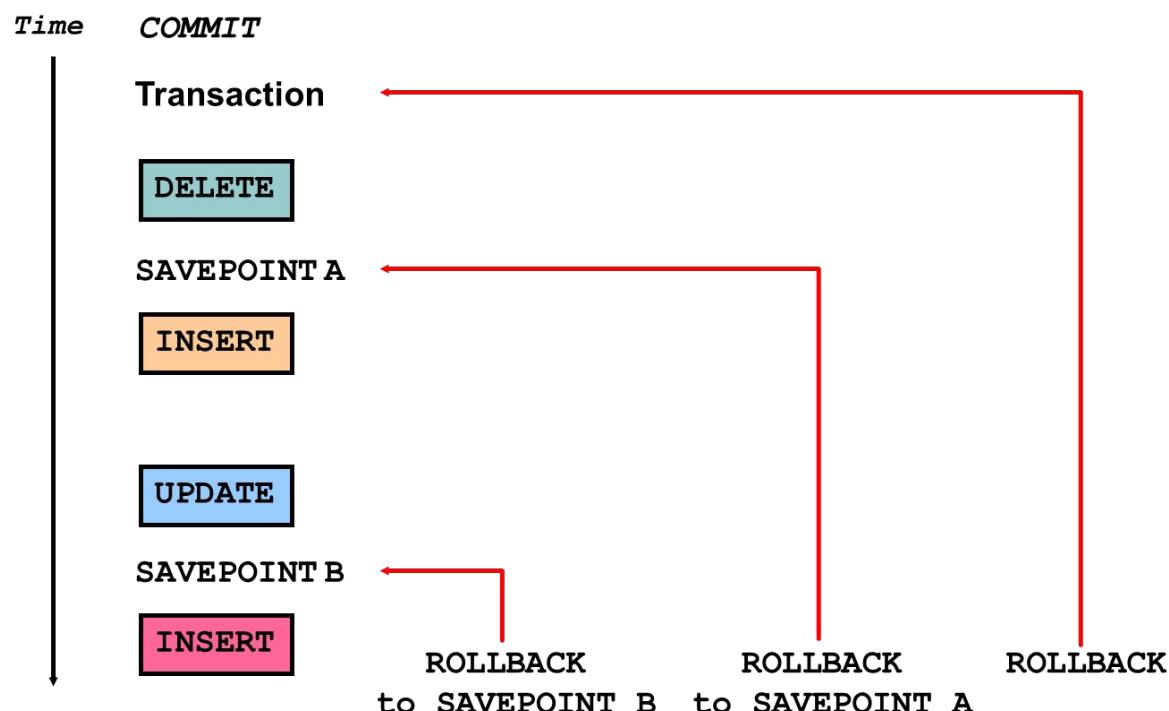
Control de Transacciones

Transacciones cumplen criterios ACID:

- Atomicity (Atomicidad)
- Consistency (Consistencia)
- Isolation (Aislamiento)
- Durability (Durabilidad)

Eventos Transacciones

- DML se ejecuta en ambiente de usuario (Aislado)
- SAVEPOINT es un punto intermedio de recuperación.
- COMMIT o ROLLBACK
- DDL o DCL realiza COMMIT automático



Devolviendo Cambios

```
UPDATE...
```

```
SAVEPOINT update_done;
```

```
SAVEPOINT update_done succeeded.
```

```
INSERT...
```

```
ROLLBACK TO update_done;
```

```
ROLLBACK TO succeeded.
```

```
DELETE FROM copy_emp;  
ROLLBACK ;
```

Devolviendo o confirmando cambios

```
DELETE FROM test;  
25,000 rows deleted.
```

```
ROLLBACK;  
Rollback complete.
```

```
DELETE FROM test WHERE id = 100;  
1 row deleted.
```

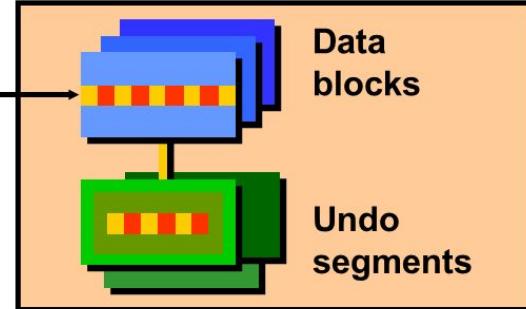
```
SELECT * FROM test WHERE id = 100;  
No rows selected.
```

```
COMMIT;  
Commit complete.
```

User A



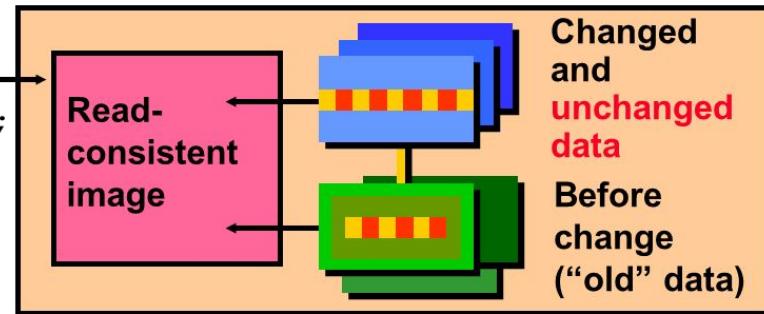
```
UPDATE employees  
SET salary = 7000  
WHERE last_name = 'Grant';
```



User B



```
SELECT *  
FROM userA.employees;
```



Implementando consistencia de lectura