

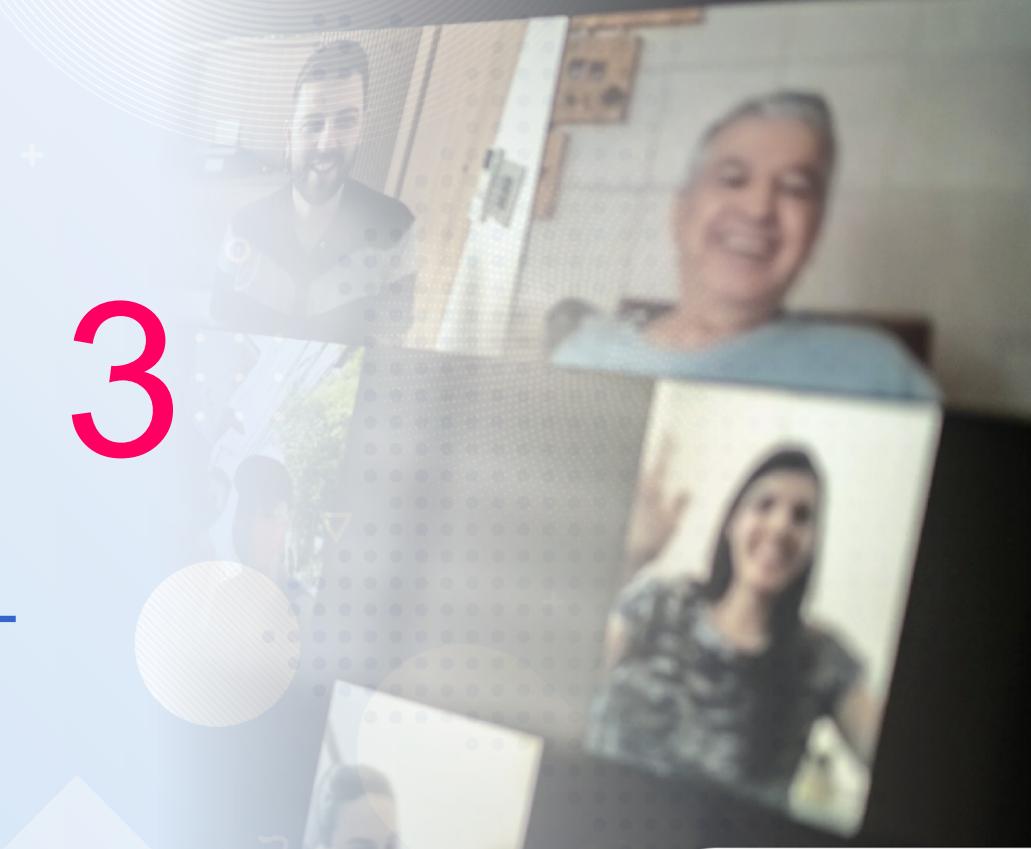


El futuro digital  
es de todos

MinTIC

# Unidad 3

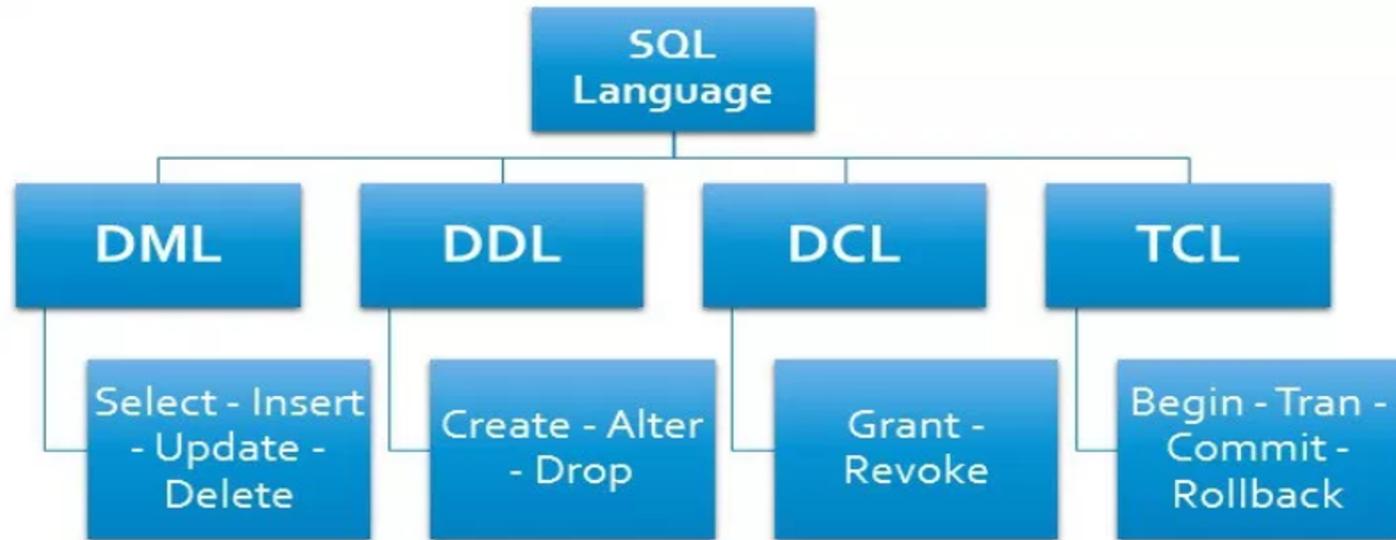
08 - SQL DDL DML





# 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



El futuro digital  
es de todos

MinTIC

# 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(PersonID)
);
```

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

```
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 SQLite3
- 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.



El futuro digital  
es de todos

MinTIC

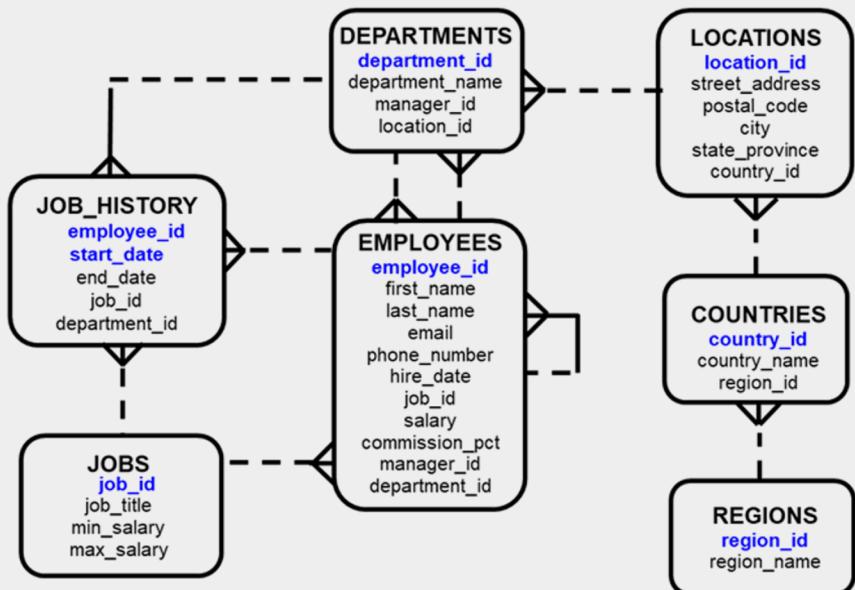
# Realizando consultas SELECT



# Modelo de Datos

Estructura base de datos de  
ejemplo

## The Human Resources (HR) Schema





Selección


Tabla 1

Proyección


Tabla 1

Enlace


Tabla 1


Tabla 2

## Capacidades de las sentencias SELECT



# Selección

```
SELECT *  
FROM departments;
```

A Z	DEPARTMENT_ID	A Z	DEPARTMENT_NAME	A Z	MANAGER_ID	A Z	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;
```

	NAME	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

- Abrir en DBeaver la base de datos HR.db
- 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.



El futuro digital  
es de todos

MinTIC

# 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
Matos	2600
Vargas	2500

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
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	Zlotkey	SA_MAN	10500
2	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
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
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

- Abrir en DBeaver la base de datos HR.db
- 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

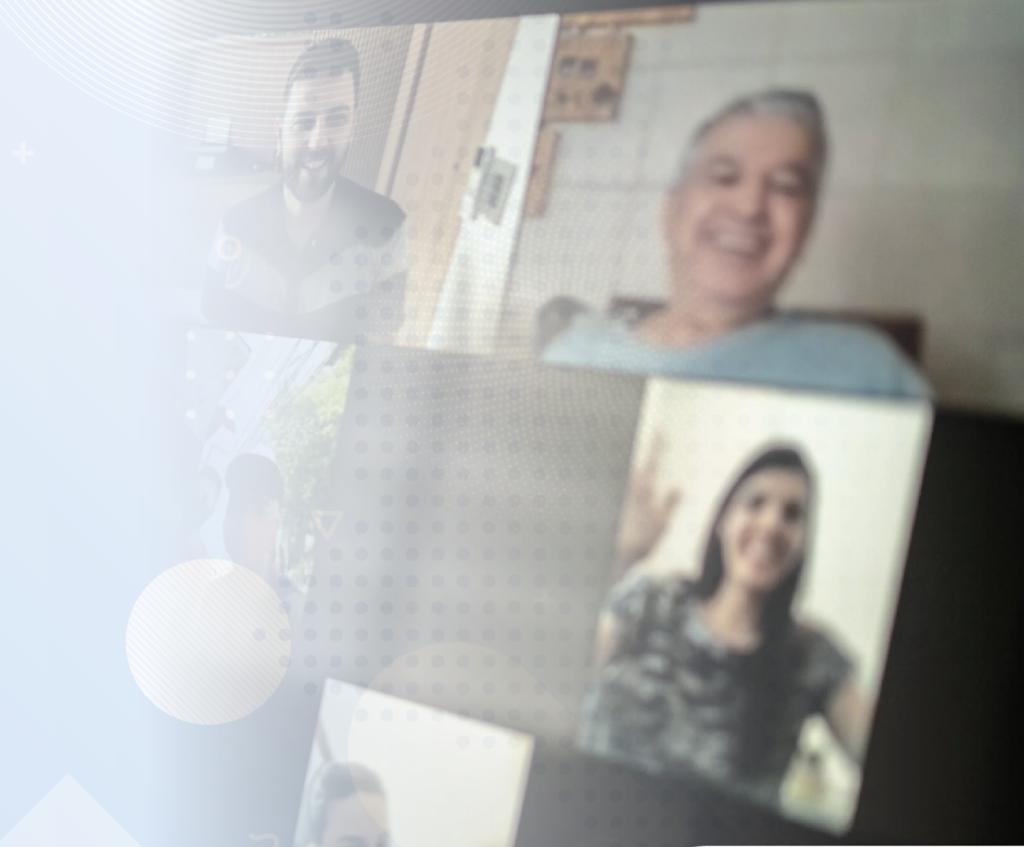
10. Muestra todos los apellidos de los empleados en los que la tercera letra del nombre es "a".
11. Muestre los apellidos de todos los empleados que tienen tanto una "a" como una "e" en su apellido.
12. Muestre el apellido, el trabajo y el salario de todos los empleados cuyos trabajos sean los de un 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%.



El futuro digital  
es de todos

MinTIC

# 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
<b>ABS (x)</b>	Devuelve el valor absoluto
<b>MOD (x, y)</b>	Devuelve el resto (módulo) de un número dividido por otro.
<b>ROUND (x, y)</b>	Redondea un número a una precisión específica
<b>CEIL (x) CEILING (x)</b>	Redondea un flotante al valor entero superior más cercano
<b>FLOOR (x)</b>	Redondea un flotante al valor entero inferior más cercano
<b>TRUNCATE (x, y) TRUNC (x, y)</b>	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 %.



# Manejo de fechas en SQLite

SQLite admite cinco funciones de fecha y hora:

1. `date(time-value, modifier, modifier,...)`
2. `time(time-value, modifier, modifier,...)`
3. `datetime(time-value, modifier, modifier,...)`
4. `julianday(time-value, modifier, modifier,...)`
5. `strftime(format, time-value, modifier, modifier,...)`

Las cinco funciones de fecha y hora toman un valor de fecha como argumento.

- La función **date** devuelve la fecha en este formato: AAAA-MM-DD.
- La función **time** devuelve la hora como HH:MM:SS.
- La función **datetime** devuelve "AAAA-MM-DD HH:MM:SS".
- La función **julianday** devuelve el número de días desde el mediodía en Greenwich el 24 de noviembre de 4714 a. C.

## Valores de Tiempo (time-value)

Un valor de tiempo puede estar en cualquiera de los siguientes formatos que se muestran a continuación.

1. YYYY-MM-DD
2. YYYY-MM-DD HH:MM
3. YYYY-MM-DD HH:MM:SS
4. YYYY-MM-DD HH:MM:SS.SSS
5. YYYY-MM-DDTHH:MM
6. YYYY-MM-DDTHH:MM:SS
7. YYYY-MM-DDTHH:MM:SS.SSS
8. HH:MM
9. HH:MM:SS
10. HH:MM:SS.SSS
11. now
12. DDDDDDDDDDD

El valor suele ser una cadena, aunque puede ser un número entero o de coma flotante en el caso del formato 12.



# Manejo de fechas en SQLite

SQLite admite cinco funciones de fecha y hora:

1. `date(time-value, modifier, modifier,...)`
2. `time(time-value, modifier, modifier,...)`
3. `datetime(time-value, modifier, modifier,...)`
4. `julianday(time-value, modifier, modifier,...)`
5. `strftime(format, time-value, modifier, modifier,...)`

Las cinco funciones de fecha y hora toman un valor de fecha como argumento.

- La función **date** devuelve la fecha en este formato: AAAA-MM-DD.
- La función **time** devuelve la hora como HH:MM:SS.
- La función **datetime** devuelve "AAAA-MM-DD HH: MM: SS".
- La función **julianday** devuelve el número de días desde el mediodía en Greenwich el 24 de noviembre de 4714 a. C.

## Modificadores (**modifier**)

El valor de la hora puede ir seguido de cero o más modificadores que alteren la fecha y / o la hora. Cada modificador es una transformación que se aplica al valor de tiempo a su izquierda. Los modificadores se aplican de izquierda a derecha; el orden es importante.

1. NNN days
2. NNN hours
3. NNN minutes
4. NNN.NNNN seconds
5. NNN months
6. NNN years
7. start of month
8. start of year
9. start of day
10. weekday N
11. unixepoch
12. localtime
13. utc



# Manejo de fechas en SQLite

SQLite admite cinco funciones de fecha y hora:

- 1.date(time-value, modifier, modifier,...)
- 2.time(time-value, modifier, modifier,...)
- 3.datetime(time-value, modifier, modifier,...)
- 4.julianday(time-value, modifier, modifier,...)
- 5.strftime(format, time-value, modifier, modifier,...)

Las cinco funciones de fecha y hora toman un valor de fecha como argumento.

- La rutina **strftime** devuelve la fecha formateada de acuerdo con la cadena de formato especificada como primer argumento.

## Cadena formateada de tiempo (**strftime**)

La cadena de formato admite las sustituciones más comunes que se encuentran en la función strftime() de la biblioteca C estándar más dos sustituciones nuevas, %f y %J.

%d	día del mes: 00
%f	fracción de segundos: SS.SSS
%H	hora: 00-24
%j	día del año: 001-366
%J	número de día Juliano
%m	mes: 01-12
%M	minutos: 00-59
%s	segundos desde 1970-01-01
%S	segundos: 00-59
%w	dia de la semana 0-6 donde Domingo es 0
%W	semana del año: 00-53
%Y	año: 0000-9999
%%	%



# Ejemplos de fechas en SQLite

Calcule la fecha actual.

```
SELECT date('now');
```

Calcule el último día del mes actual.

```
SELECT date('now', 'start of month', '+1 month', '-1 day');
```

Calcule la fecha y la hora con una marca de tiempo Unix 1092941466.

```
SELECT datetime(1092941466, 'unixepoch');
```

Calcule la fecha y la hora con una marca de tiempo Unix 1092941466

y compense su zona horaria local.

```
SELECT datetime(1092941466, 'unixepoch', 'localtime');
```

Calcule la marca de tiempo actual de Unix.

```
SELECT strftime('%s', 'now');
```

Calcule el número de días desde la firma de la Declaración de Independencia de EE. UU.

```
SELECT julianday('now') - julianday('1776-07-04');
```

Calcule la cantidad de segundos desde un momento particular en 2004:

```
SELECT strftime('%s', 'now')  
      - strftime('%s', '2004-01-01 02:34:56');
```

Calcule la fecha del primer martes de octubre para el año actual.

```
SELECT date('now', 'start of year', '+9 months',  
           'weekday 2');
```

Calcule el tiempo desde la época de Unix en segundos (como strftime ('% s', 'ahora') excepto que incluye parte fraccionaria):

```
SELECT (julianday('now') - 2440587.5)*86400.0;
```



# 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

- Abrir en DBeaver la base de datos HR.db
- 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



# Para la próxima sesión...

- Terminar los ejercicios que no se terminaron... (si aplica)
- Instalación de SQLite 3
- Ver los videos:
  - Curso Video Base de Datos (el resto de los videos)