

---

# SQL

# Introducción

---

- ◆ Un sistema de gestión de bases de datos relacional (RDBMS) es un programa que permite **crear, actualizar, administrar** una base de datos relacional y **proporciona un acceso controlado** a la base de datos.
- ◆ SGBDR (sistema gestor de bases de datos relacionales)
- ◆ La mayoría de los RDBMS comerciales utilizan el lenguaje de consultas estructuradas (**SQL**) para acceder a la base de datos.
- ◆ Los principales productos RDBMS son Oracle, **DB2** de IBM y **Microsoft SQL Server**.

# Introducción

---

- ◆ Una **base de datos relacional** es una colección de elementos de datos organizados en un conjunto de tablas formalmente descritas desde la que se puede acceder a los datos o volver a montarlos de muchas maneras diferentes sin tener que reorganizar las tablas de la base.
- ◆ La base de datos relacional fue inventada por E.F. Codd en IBM en 1970.

# Introducción

---

- ◆ Los SGBD permiten definir **subesquemas o vistas**: cada usuario “ve” la parte de la base de datos que necesita.
- ◆ Además:
  - ◆ Las vistas proporcionan un nivel de seguridad adicional.
  - ◆ Las vistas proporcionan un mecanismo para que los usuarios vean los datos en el formato deseado.
  - ◆ Las vistas representan una imagen consistente y permanente de la base de datos, incluso si ésta cambia su estructura.

# Introducción

---

- ◆ **Papeles en el entorno de las bases de datos:**
- ◆ El **administrador de la base de datos** realiza las siguientes tareas:
  - Diseño físico e implementación de la base de datos.
  - Control de la seguridad y la concurrencia.
  - Mantenimiento del sistema.
  - Mantenimiento de las prestaciones.
- ◆ Los **diseñadores de la base de datos** llevan a cabo el diseño lógico de la base de datos.
- ◆ Los **programadores de aplicaciones** implementan los programas que darán servicio a los usuarios finales.
- ◆ Los **usuarios finales** son los clientes de la base de datos.

# Introducción

---

- ◆ La interfaz estándar de programa de usuario y aplicación a una base de datos relacional es el lenguaje de consultas estructuradas (**SQL**).
- ◆ Una base de datos relacional tiene la importante ventaja de ser **fácil de extender**:
- ◆ Después de la creación original de una base de datos, una nueva categoría de datos se puede añadir sin necesidad de que todas las aplicaciones existentes sean modificadas.

# Introducción

---

- ◆ Una base de datos relacional es un conjunto de tablas que contienen datos provistos en categorías predefinidas.
- ◆ Cada tabla (que a veces se llaman ‘relación’) contiene una o más categorías de datos en columnas.
- ◆ Cada fila contiene una instancia única de datos para las categorías definidas por las columnas.

# Introducción

---

- ◆ **Ejemplo**, una base de datos típica de ingreso de solicitudes de negocio incluiría una tabla que describiera a un cliente con columnas para el nombre, dirección, número de teléfono, y así sucesivamente.
- ◆ Otra tabla identificaría el pedido: producto, cliente, fecha, precio de venta, y así sucesivamente.
- ◆ Un usuario de la base de datos podría obtener una vista de la base de datos que se ajuste a sus necesidades.
- ◆ Por ejemplo, **un gerente de sucursal** podría preferir una vista o informe sobre todos los clientes que han comprado productos después de una fecha determinada. Un gerente de servicios financieros en la misma empresa podría, desde las mismas tablas, obtener un informe sobre las cuentas que deben ser pagadas.



# Introducción

---

- ◆ Al crear una base de datos relacional, se puede definir el dominio de posibles valores de una columna de datos y restricciones adicionales que pueden aplicarse a ese valor de dato.
- ◆ La definición de una base de datos relacional resulta en una tabla de metadatos o descripciones formales de las tablas, columnas, dominios y restricciones.

# Introducción

---

- ◆ La relación entre una tabla padre y un hijo se lleva a cabo por medio de las claves primarias y **claves foráneas o ajenas**.
- ◆ Las **claves primarias** son la clave principal de un registro dentro de una tabla y estas **deben cumplir con la integridad de datos**.
- ◆ Las **claves ajenas** se colocan en la tabla hija, contienen el mismo valor que la clave primaria del registro padre **Por** medio de estas se hacen las formas relacionales.

# Introducción

---

- ◆ La **integridad de datos** se refiere la corrección y completitud de la información en una base de datos.
- ◆ Cuando los contenidos se modifican con sentencias insertando, borrando o actualizando, la integridad de los datos almacenados puede perderse de muchas maneras diferentes.

# Tipos de restricciones de integridad

---

- ◆ **Integridad de dominio**
- ◆ **Integridad de entidad**
- ◆ **Integridad referencial**

# Tipos de restricciones de integridad

---

- ◆ **Integridad de dominio:**
- ◆ **Datos Requeridos:** establece que una columna tenga un valor no NULL. Se define efectuando la declaración de una columna es NOT NULL cuando la tabla que contiene las columnas se crea por primera vez, como parte de la sentencia CREATE TABLE.
- ◆ **Chequeo de Validez:** cuando se crea una tabla cada columna tiene un tipo de datos y el DBMS asegura que solamente los datos del tipo especificado sean ingresados en la tabla.

# Tipos de restricciones de integridad

---

- ◆ **Integridad de entidad:**

- ◆ establece que la clave primaria de una tabla debe tener un valor único para cada fila de la tabla; si no, la base de datos perderá su integridad.
- ◆ Se especifica en la sentencia `CREATE TABLE`. El DBMS comprueba automáticamente la unicidad del valor de la clave primaria con cada sentencia `INSERT` Y `UPDATE`.
- ◆ Un intento de insertar o actualizar una fila con un valor de la clave primaria ya existente fallará.

# Tipos de restricciones de integridad

---

## ◆ Integridad referencial:

- ◆ asegura la integridad entre las **claves foráneas y primarias** (relaciones padre/hijo).
- ◆ Existen cuatro actualizaciones de la base de datos que pueden corromper la integridad referencial:
  1. La inserción de una fila hijo se produce cuando no coincide la clave foránea con la clave primaria del padre.
  2. La actualización en la clave foránea de la fila hijo, donde se produce una actualización en la clave ajena de la fila hijo con una sentencia UPDATE y la misma no coincide con ninguna llave primaria.
  3. La supresión de una fila padre, con la que, si una fila padre -que tiene uno o más hijos- se suprime, las filas hijos quedarán huérfanas.
  4. La actualización de la clave primaria de una fila padre, donde si en una fila padre, que tiene uno o más hijos se actualiza su clave primaria, las filas hijos quedarán huérfanas.

# Normalización

---

- ◆ **Normalización:**

- ◆ es el proceso de organizar los datos de una base de datos. Se incluye la creación de tablas y el establecimiento de relaciones entre ellas según reglas diseñadas tanto para proteger los datos como para hacer que la base de datos sea más flexible al eliminar la redundancia y las dependencias incoherentes.
- ◆ Las bases de datos relacionales se normalizan para:
- ◆ Evitar la redundancia de los datos.
- ◆ Evitar problemas de actualización de los datos en las tablas.
- ◆ Proteger la integridad de los datos.



# Primera forma normal. 1FN

---

- ◆ **Primera forma normal**
- ◆ Elimine los grupos repetidos de las tablas individuales.
- ◆ Cree una tabla independiente para cada conjunto de datos relacionados.
- ◆ Identifique cada conjunto de datos relacionados con una **clave principal**.

# Segunda forma normal. 2FN

---

- ◆ **Segunda forma normal**
- ◆ Cree tablas independientes para conjuntos de valores que se apliquen a varios registros.
- ◆ Relacione estas tablas con una clave externa.

# Segunda forma normal. 2FN

---

- ◆ La regla de la Segunda Forma Normal (2FN) establece que todas las dependencias parciales se deben eliminar y separar dentro de sus propias tablas.
- ◆ Una dependencia parcial es un término que describe a aquellos datos que no dependen de la clave de la tabla para identificarlos.

# Tercera Forma Normal. 3FN

---

- ◆ **Tercera Forma Normal**
- ◆ Ningún atributo no-primario de la tabla es dependiente transitivamente de una clave primaria.
  - Un atributo no-primario es un atributo que no pertenece a ninguna clave candidata.

# Introducción a SQL

---

# El lenguaje de programación SQL

---

- ◆ Las siglas **SQL** provienen del inglés Structured Query Language, Lenguaje Estructurado de Consultas.
- ◆ Este lenguaje es el específico que se usa para consulta de datos en el Sistema Gestor de Base de Datos Relacionales (SGBDR).

# El lenguaje de programación SQL

---

- ◆ Lenguaje **declarativo** de acceso a bases de datos que combina construcciones del álgebra relacional y el cálculo relacional.
- ◆ Originalmente desarrollado en los '70 por IBM en su Research Laboratory de San José a partir del cálculo de predicados creado por **Codd**.
- ◆ Lenguaje estándar *de facto* en los SGBD comerciales

# El lenguaje de programación SQL

---

## ◆ Estándares:

- SEQUEL(Structured English QUery Language), IBM 1976
- SQL-86 (ANSI SQL)
- SQL-89 (SQL1)
- SQL-92 (SQL2), gran revisión del estándar
- SQL:1999 (SQL3), Añade disparadores, algo de OO, ...
- SQL:2003. Añade XML, secuencias y columnas autonuméricas.



# Características de SQL

---

- ◆ Tipos de declaraciones SQL
- ◆ Definición de vistas
- ◆ Autorización
  - Definición de usuarios y privilegios
- ◆ Integridad de datos
- ◆ Control de Transacciones

# Tipos de declaraciones SQL

---

◆ Las sentencias de SQL se clasifican en cuatro tipos:

1. **El Language de Definición de Datos /**  
**DDL - Data Definition Language**
2. **El Lenguaje de Manipulación de Datos /**  
**DML - Data Manipulation Language**
3. **El Lenguaje de Control de Datos /**  
**DCL - Data Control Language**
4. **El lenguaje de Control de Transacciones /**  
**TCL - Transaction Control Language**

- 
- ◆ Con el uso de estas sentencias podemos:
  - ◆ definir la estructura de una base de datos mediante la creación y la modificación de los objetos de base de datos
  - ◆ manipular los datos de una tabla a través de actualizaciones o supresiones
  - ◆ controlar qué usuarios pueden leer / escribir datos
  - ◆ gestionar transacciones para crear una única unidad de trabajo.

# Tipos de declaraciones SQL

---

## 1.El Language de Definición de Datos /

### **DDL - Data Definition Language**

- Proporciona comandos para la creación, borrado y modificación de esquemas relacionales

## 2.El Lenguaje de Manipulación de Datos /

### **DML - Data Manipulation Language**

- Basado en el álgebra relacional y el cálculo relacional permite realizar consultas y adicionalmente insertar, borrar y actualizar de tuplas
- Ejecutado en una consola interactiva
- Embebido dentro de un lenguaje de programación de propósito general

# Tipos de declaraciones SQL

---

## 3.El Lenguaje de Control de Datos /

### **DCL - Data Control Language**

- Permiten al administrador controlar el acceso a los datos contenidos en la Base de Datos.

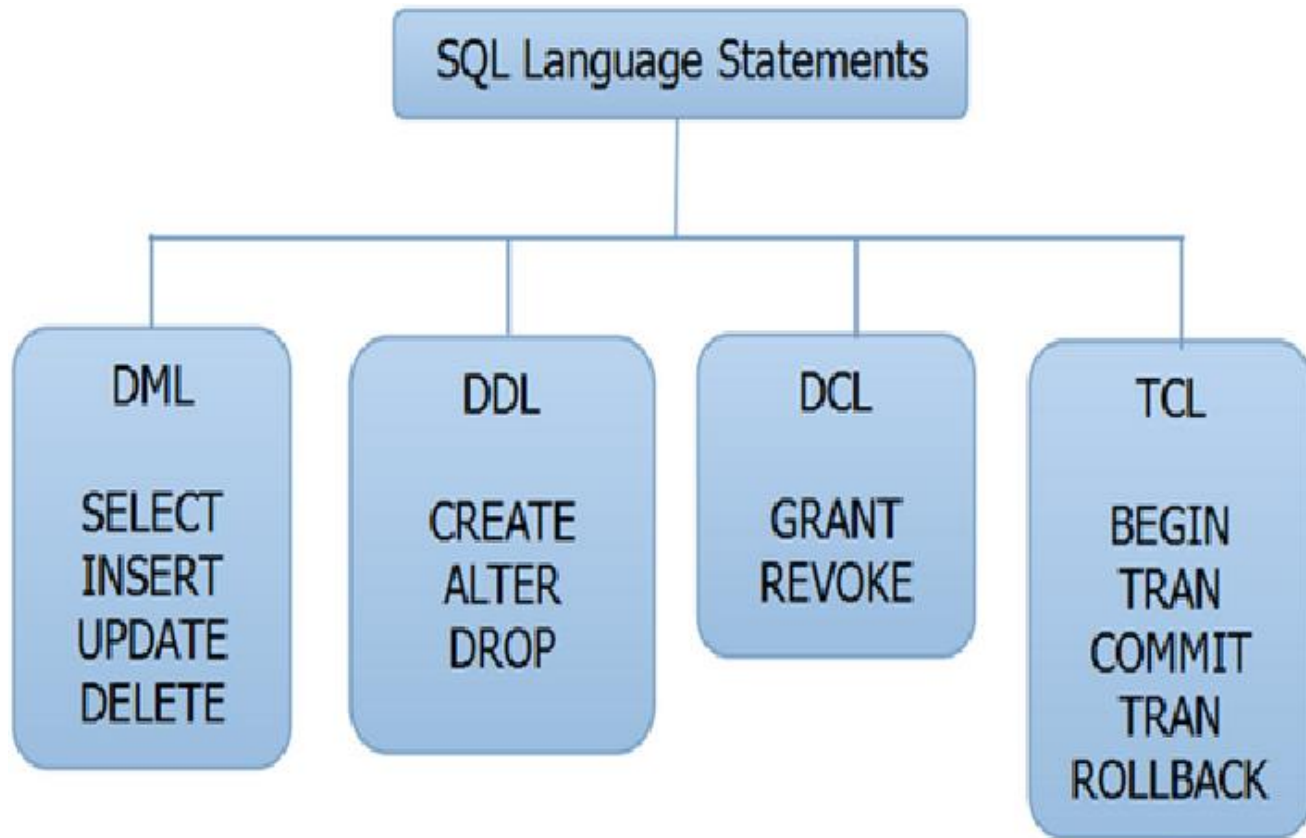
## 4.El lenguaje de Control de Transacciones /

### **TCL - Transaction Control Language**

- Permiten controlar y gestionar transacciones para mantener la integridad de los datos

# Tipos de declaraciones SQL

---



# SQL - DDL

---

◆CREATE

◆ALTER

◆DROP

# Columna autoincrementada

---

```
◆CREATE TABLE Tabla (  
campo1  INTEGER IDENTITY(1,1) NOT  
        NULL,  
campo2  INTEGER NOT NULL,  
        campo3  INTEGER,  
campo4  VARCHAR(6),  
CONSTRAINT PK_Habitaciones PRIMARY  
        KEY (idHabitacion));
```



# Creación de tablas

- ◆ La creación de tablas se lleva a cabo con la sentencia **CREATE TABLE**.

- ◆ **Ejemplo:** creación del siguiente esquema de BD.

CLIENTES (DNI, NOMBRE, DIR)    SUCURSALES (NSUC, CIUDAD)  
 CUENTAS (COD, DNI, NSUCURS, SALDO)

- ◆ Se empieza por las tablas más independientes:

```
CREATE TABLE CLIENTES (  
    DNI VARCHAR(9) NOT NULL,  
    NOMBRE VARCHAR(20),  
    DIR VARCHAR(30),  
    PRIMARY KEY (DNI)  
);
```

```
CREATE TABLE SUCURSALES (  
    NSUC VARCHAR(4) NOT NULL,  
    CIUDAD VARCHAR(30),  
    PRIMARY KEY (NSUC)  
);
```

# Creación de Tablas (cont.)

---

- ◆ El siguiente paso es crear la tabla CUENTAS, con las claves externas:

```
CREATE TABLE CUENTAS (  
    COD VARCHAR(4) NOT NULL,  
    DNI VARCHAR(9) NOT NULL,  
    NSUCURS VARCHAR(4) NOT NULL,  
    SALDO INT DEFAULT 0,  
    PRIMARY KEY (COD, DNI, NSUCURS),  
    FOREIGN KEY (DNI) REFERENCES CLIENTES (DNI),  
    FOREIGN KEY (NSUCURS) REFERENCES SUCURSALES (NSUC) );
```

◆ .

# Modificación y eliminación de tablas

---

## ◆ Modificación de tablas: sentencia **ALTER TABLE**.

- Es posible añadir, modificar y eliminar campos. Ejemplos:
  - Adición del campo PAIS a la tabla CLIENTES
- **ALTER TABLE** CLIENTES **ADD** PAIS **VARCHAR**(10) ;
  - Modificación del tipo del campo PAIS
- **ALTER TABLE** CLIENTES **MODIFY** PAIS **VARCHAR**(20) ;
  - Eliminación del campo PAIS de la tabla CLIENTES
- **ALTER TABLE** CLIENTES **DROP** PAIS ;
- También es posible añadir nuevas restricciones a la tabla (claves externas, restricciones check).

## ◆ Eliminación de tablas: sentencia **DROP TABLE**.

- **DROP TABLE** CUENTAS ;
  - Las tablas a las que referencia deben haber sido eliminadas antes.

# Ejemplo

---

```
CREATE TABLE EMPLOYEE
(EMPNO CHARACTER(6) PRIMARY KEY
,FIRSTNME VARCHAR(12) NOT NULL
,MIDINIT CHARACTER(1)
,LASTNAME VARCHAR(15) NOT NULL
,WORKDEPT CHARACTER(3)
,PHONENO CHARACTER(4)
,HIREDATE DATE
,JOB CHARACTER(8)
,EDLEVEL SMALLINT NOT NULL
,SEX CHARACTER(1)
,BIRTHDATE DATE
,SALARY DECIMAL(9,2)
,BONUS DECIMAL(9,2)
,COMM DECIMAL(9,2))
```

# Ejemplo

EMPNO	FIRSTNME	M	LASTNAME	DPT	PH#	HIREDATE	SX	ED	BIRTHDATE	SALARY	COMM
000010	CHRISTINE	I	HAAS	A00	3978	1995-01-01	F	18	1963-08-24	152750	4220
000020	MICHAEL	L	THOMPSON	B01	3476	2003-10-10	M	18	1978-02-02	94250	3300
000030	SALLY	A	KWAN	C01	4738	2005-04-05	F	20	1971-05-11	98250	3060
000050	JOHN	B	GEYER	E01	6789	1979-08-17	M	16	1955-09-15	80175	3214
000060	IRVING	F	STERN	D11	6423	2003-09-14	M	16	1975-07-07	72250	2580
000070	EVA	D	PULASKI	D21	7831	2005-09-30	F	16	2003-05-26	96170	2893
000090	EILEEN	W	HENDERSON	E11	5498	2000-08-15	F	16	1971-05-15	89750	2380
000100	THEODORE	Q	SPENSER	E21	0972	2000-06-19	M	14	1980-12-18	86150	2092
000110	VINCENZO	G	LUCCHESSI	A00	3490	1988-05-16	M	19	1959-11-05	66500	3720
000120	SEAN		O'CONNELL	A00	2167	1993-12-05	M	14	1972-10-18	49250	2340
000130	DELORES	M	QUINTANA	C01	4578	2001-07-28	F	16	1955-09-15	73800	1904
000140	HEATHER	A	NICHOLLS	C01	1793	2006-12-15	F	18	1976-01-19	68420	2274
000150	BRUCE		ADAMSON	D11	4510	2002-02-12	M	16	1977-05-17	55280	2022
000160	ELIZABETH	R	PIANKA	D11	3782	2006-10-11	F	17	1980-04-12	62250	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1999-09-15	M	16	1981-01-05	44680	1974
000180	MARILYN	S	SCOUTTEN	D11	1682	2003-07-07	F	17	1979-02-21	51340	1707
000190	JAMES	H	WALKER	D11	2986	2004-07-26	M	16	1982-06-25	50450	1636
000200	DAVID		BROWN	D11	4501	2002-03-03	M	16	1971-05-29	57740	2217
000210	WILLIAM	T	JONES	D11	0942	1998-04-11	M	17	2003-02-23	68270	1462
000220	JENNIFER	K	LUTZ	D11	0672	1998-08-29	F	18	1978-03-19	49840	2387
000230	JAMES	J	JEFFERSON	D21	2094	1996-11-21	M	14	1980-05-30	42180	1774
000240	SALVATORE	M	MARINO	D21	3780	2004-12-05	M	17	2002-03-31	48760	2301
000250	DANIEL	S	SMITH	D21	0961	1999-10-30	M	15	1969-11-12	49180	1534
000260	SYBIL	P	JOHNSON	D21	8953	2005-09-11	F	16	1976-10-05	47250	1380
000270	MARIA	L	PEREZ	D21	9001	2006-09-30	F	15	2003-05-26	37380	2190
000280	ETHEL	R	SCHNEIDER	E11	8997	1997-03-24	F	17	1976-03-28	36250	2100
000290	JOHN	R	PARKER	E11	4502	2006-05-30	M	12	1985-07-09	35340	1227

# Ejemplo

- ◆ DEPARTMENT(DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION)

```
CREATE TABLE DEPARTMENT
(DEPTNO CHARACTER(3) PRIMARY KEY
,DEPTNAME VARCHAR(36) NOT NULL
,MGRNO CHARACTER(6)
,ADMRDEPT CHARACTER(3) NOT NULL
,LOCATION CHARACTER(16)) ;
```

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	-
B01	PLANNING	000020	A00	-
C01	INFORMATION CENTER	000030	A00	-
D01	DEVELOPMENT CENTER	-	A00	-
D11	MANUFACTURING SYSTEMS	000060	D01	-
D21	ADMINISTRATION SYSTEMS	000070	D01	-
E01	SUPPORT SERVICES	000050	A00	-
E11	OPERATIONS	000090	E01	-
E21	SOFTWARE SUPPORT	000100	E01	-
F22	BRANCH OFFICE F2	-	E01	-
G22	BRANCH OFFICE G2	-	E01	-

# SQL - LMD

---

## ◆ Selección:

- SELECT

## ◆ Modificación:

- INSERT
- UPDATE
- DELETE

# Estructura de la sentencia SELECT

---

**SELECT** A1, ..., An

-Describe la salida deseada con:

- Nombres de columnas
- Expresiones aritméticas
- Literales
- Funciones escalares
- Funciones de columna

**FROM** T1, ..., Tn

- Nombres de las tablas / vistas

**WHERE** P

- Condiciones de selección de filas

**GROUP BY** A<sub>i1</sub>, ..., A<sub>in</sub>

- Nombre de las columnas

**HAVING** Q

- Condiciones de selección de grupo

**ORDER BY** A<sub>j1</sub>, ..., A<sub>jn</sub>

- Nombres de columnas



# Estructura básica de la sentencia SELECT

---

## ◆ Consta de tres cláusulas:

### ■ **SELECT**

- La lista de los atributos que se incluirán en el resultado de una consulta.

### ■ **FROM**

- Especifica las relaciones que se van a usar como origen en el proceso de la consulta.

### ■ **WHERE**

- Especifica la condición de filtro sobre las tuplas en términos de los atributos de las relaciones de la cláusula **FROM**.

# Estructura básica de la sentencia SELECT

---

## ◆ Una consulta SQL tiene la forma:

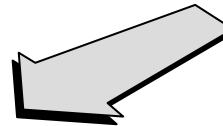
**SELECT**  $A_1, \dots, A_n$       /\* Lista de atributos \*/  
**FROM**  $R_1, \dots, R_m$       /\* Lista de relaciones. A veces opcional \*/  
**WHERE**  $P$ ;      /\* Condición. Cláusula opcional \*/

- Es posible que exista el mismo nombre de atributo en dos relaciones distintas.
- Se añade "*NOMBRE\_RELACION*." antes del nombre para desambiguar.

# Proyección de algunas columnas

---

```
SELECT DEPTNO, DEPTNAME, ADMRDEPT  
FROM DEPARTMENT
```



DEPTNO	DEPTNAME	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	A00
B01	PLANNING	A00
C01	INFORMATION CENTER	A00
D01	DEVELOPMENTCENTER	A00
D11	MANUFACTURING SYSTEMS	D01
D21	ADMINISTRATION SYSTEMS	D01
E01	SUPPORT SERVICES	A00
E11	OPERATIONS	E01
E21	SOFTWARE SUPPORT	E01

# Eliminación de filas duplicadas

---

- ◆ SQL permite duplicados en el resultado
- ◆ Para eliminar las tuplas repetidas se utiliza la cláusula **DISTINCT**.
- ◆ También es posible pedir explícitamente la inclusión de filas repetidas mediante el uso de la cláusula **ALL**.

```
SELECT ADMRDEPT  
FROM DEPARTMENT
```



ADMRDEPT

A00  
A00  
A00  
A00  
D01  
D01  
A00  
E01  
E01

```
SELECT ALL ADMRDEPT  
FROM DEPARTMENT
```

```
SELECT DISTINCT ADMRDEPT  
FROM DEPARTMENT
```

ADMRDEPT

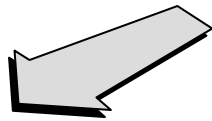
A00  
D01  
E01

# Eliminación de filas duplicadas

---

◆ ¿Qué trabajos realiza cada departamento?

```
SELECT  DISTINCT WORKDEPT, JOB  
FROM EMPLOYEE
```

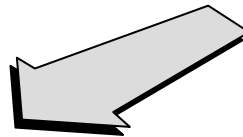


WORKDEPT	JOB
A00	CLERK
A00	PRES
A00	SALESREP
B01	MANAGER
C01	ANALYST
C01	MANAGER
D11	DESIGNER
D11	MANAGER
D21	CLERK
D21	MANAGER
E01	MANAGER
E11	MANAGER
E11	OPERATOR
E21	FIELDREP
E21	MANAGER

# Proyección de todos los atributos

- ◆ Se puede pedir la proyección de todos los atributos de la consulta mediante utilizando el símbolo '\*'
  - La tabla resultante contendrá todos los atributos de las tablas que aparecen en la cláusula **FROM**.

```
SELECT * FROM DEPARTMENT
```



DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	
B01	PLANNING	000020	A00	
C01	INFORMATION CENTER	000030	A00	
D01	DEVELOPMENTCENTER	- - - - -	A00	
D11	MANUFACTURING SYSTEMS	000060	D01	
D21	ADMINISTRATION SYSTEMS	000070	D01	
E01	SUPPORT SERVICES	000050	A00	
E11	OPERATIONS	000090	E01	
E21	SOFTWARE SUPPORT	000100	E01	

# Salida ordenada

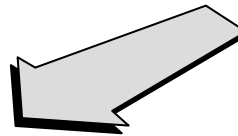
---

- ◆ SQL permite controlar el orden en el que se presentan las tuplas de una relación mediante la cláusula **ORDER BY**.
- ◆ La cláusula **ORDER BY** tiene la forma  
**ORDER BY *A1* <DIRECCION>, ..., *An* <DIRECCION>**
  - *A1*, ..., *An* son atributos de la relación resultante de la consulta
  - *Ai* <DIRECCION> controla si la ordenación es Ascendente '**ASC**' o descendente '**DESC**' por el campo *Ai*. Por defecto la ordenación se realiza de manera ascendente.
- ◆ La ordenación se realiza tras haber ejecutado la consulta sobre las tuplas resultantes.
- ◆ La ordenación puede convertirse en una operación costosa dependiendo del tamaño de la relación resultante.

## Salida ordenada (cont.)

---

```
SELECT  DEPTNO, DEPTNAME, ADMRDEPT
FROM DEPARTMENT
ORDER BY ADMRDEPT ASC
```



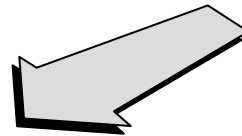
DEPTNO DEPTNAME		ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	A00
C01	INFORMATION CENTER	A00
B01	PLANNING	A00
E01	SUPPORTSERVICES	A00
D01	DEVELOPMENTCENTER	A00
D11	MANUFACTURING SYSTEMS	D01
D21	ADMINISTRATION SYSTEMS	D01
E21	SOFTWARE SUPPORT	E01
E11	OPERATIONS	E01



## Salida ordenada (cont.)

---

```
SELECT  DEPTNO, DEPTNAME, ADMRDEPT
FROM    DEPARTMENT
ORDER BY ADMRDEPT ASC, DEPTNO DESC
```



ADMRDEPT DEPTNAME		DEPTNO
A00	SUPPORT SERVICES	E01
A00	DEVELOPMENT CENTER	D01
A00	INFORMATION CENTER	C01
A00	PLANNING	B01
A00	SPIFFY COMPUTER SERVICE DIV.	A00
D01	ADMINISTRATION SYSTEMS	D21
D01	MANUFACTURING SYSTEMS	D11
E01	SOFTWARE SUPPORT	E21
E01	OPERATIONS	E11

# Selección de filas

---

- ◆ La cláusula WHERE permite filtrar las filas de la relación resultante.
  - La condición de filtrado se especifica como un predicado.
- ◆ El predicado de la cláusula WHERE puede ser simple o complejo
  - Se utilizan los conectores lógicos AND (conjunción), OR (disyunción) y NOT (negación)
- ◆ Las expresiones pueden contener
  - Predicados de comparación
  - BETWEEN / NOT BETWEEN
  - IN / NOT IN (con y sin subconsultas)
  - LIKE / NOT LIKE
  - IS NULL / IS NOT NULL
  - ALL, SOME/ANY (subconsultas)
  - EXISTS (subconsultas)

# Selección de filas (cont.)

---

## ◆ Predicados de comparación

- Operadores: =, <> (es el  $\neq$ ), <, <=, >=, >

## ◆ **BETWEEN Op1 AND Op2**

- Es el operador de comparación para intervalos de valores o fechas.

## ◆ **IN** es el operador que permite comprobar si un valor se encuentra en un conjunto.

- Puede especificarse un conjunto de valores (**Val1, Val2, ...**)
- Puede utilizarse el resultado de otra consulta **SELECT**.

# Selección de filas (cont.)

---

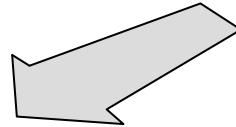
- ◆ **LIKE** es el operador de comparación de cadenas de caracteres.
  - SQL distingue entre mayúsculas y minúsculas
  - Las cadenas de caracteres se incluyen entre comillas simples
  - SQL permite definir **patrones** a través de los siguientes caracteres:
    - '%', que es equivalente a "cualquier subcadena de caracteres"
    - '\_', que es equivalente a "cualquier carácter"
- ◆ **IS NULL** es el operador de comparación de valores nulos.

# Ejemplo de selección de filas

---

- ◆ ¿Qué departamentos informan al A00?

```
SELECT  DEPTNO, ADMRDEPT  
FROM    DEPARTMENT  
WHERE  ADMRDEPT='A00'
```



<u>DEPTNO</u>	<u>ADMRDEPT</u>
A00	A00
B01	A00
C01	A00
D01	A00
E01	A00

# Ejemplo de selección de filas

---

- ◆ Necesito el apellido y el nivel de formación de los empleados cuyo nivel de formación es mayor o igual a 19

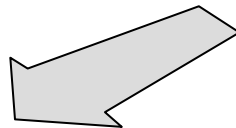
```
SELECT LASTNAME, EDLEVEL  
FROM EMPLOYEE  
WHERE EDLEVEL >= 19
```

# Ejemplo de selección de filas

---

- ◆ Necesito el número de empleado, apellido y fecha de nacimiento de aquellos que hayan nacido después del 1 de enero de 1955 (inclusive).

```
SELECT EMPNO, LASTNAME, BIRTHDATE  
FROM EMPLOYEE  
WHERE BIRTHDATE >='1955-01-01'  
ORDER BY BIRTHDATE
```



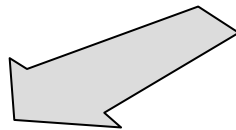
EMPNO	LASTNAME	BIRTHDATE
000160	PIANKA	1955-04-12
000100	SPENCER	1956-12-18

# Múltiples condiciones - AND

---

- ◆ Necesito el número de empleado, el trabajo y el nivel de formación de los analistas con un nivel de educación 16

```
SELECT EMPNO, JOB, EDLEVEL  
FROM EMPLOYEE  
WHERE JOB='ANALYST' AND EDLEVEL=16
```



EMPNO	JOB	EDLEVEL
000130	ANALYST	16



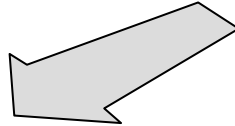
# Múltiples condiciones – AND/OR

- ◆ Obtener el número de empleado, el trabajo y el nivel de formación de todos los analistas con un nivel 16 y de todos los empleados de nivel 18. La salida se ordena por trabajo y nivel

```
SELECT EMPNO, JOB, EDLEVEL  
FROM EMPLOYEE
```

```
WHERE (JOB='ANALYST' AND EDLEVEL=16)  
      OR EDLEVEL=18
```

```
ORDER BY JOB, EDLEVEL
```



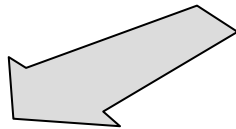
EMPNO	JOB	EDLEVEL
000130	ANALYST	16
000140	ANALYST	18
000220	DESIGNER	18
000020	MANAGER	18
000010	PRES	18

# SELECT con BETWEEN

---

- ◆ Obtener el número de empleado y el nivel de todos los empleados con un nivel entre 12 y 15

```
SELECT EMPNO, EDLEVEL  
FROM EMPLOYEE  
WHERE EDLEVEL BETWEEN 12 AND 15  
ORDER BY EDLEVEL
```



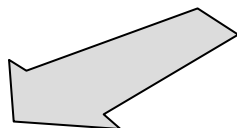
EMPNO	EDLEVEL
000310	12
000290	12
000300	14
000330	14
000100	14
000230	14
000120	14
000270	15
000250	15

# SELECT con IN

- ◆ Listar los apellidos y nivel de formación de todos los empleados de nivel 14, 19 o 20.

- El resultado clasificado por nivel y apellido

```
SELECT LASTNAME, EDLEVEL
FROM EMPLOYEE
WHERE EDLEVEL IN (14, 19, 20)
ORDER BY EDLEVEL, LASTNAME
```



<u>LASTNAME</u>	<u>EDLEVEL</u>
JEFFERSON	14
LEE	14
O'CONNELL	14
SMITH	14
SPENSER	14
LUCCHESI	19
KWAN	20

# Búsqueda parcial - LIKE

---

- ◆ Obtener el apellido de todos los empleados cuyo apellido empiece por G

```
SELECT LASTNAME  
FROM EMPLOYEE  
WHERE LASTNAME LIKE 'G%' ;
```

# Búsqueda parcial – LIKE

## Ejemplos con %

---

```
SELECT LASTNAME  
FROM EMPLOYEE  
WHERE LASTNAME LIKE ' %SON ' ;
```

THOMPSON  
HENDERSON  
ADAMSON  
JEFFERSON  
JOHNSON

```
SELECT LASTNAME  
FROM EMPLOYEE  
WHERE LASTNAME LIKE ' %M%N% ' ;
```

THOMPSON  
ADAMSON  
MARINO

# Búsqueda parcial – LIKE

## Ejemplos con \_

---

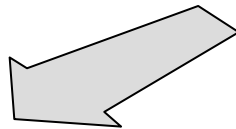
- ◆ ¿Qué empleados tienen una C como segunda letra de su apellido?

```
SELECT LASTNAME  
FROM EMPLOYEE  
WHERE LASTNAME LIKE '_C%' ;
```

# Búsqueda parcial – NOT LIKE

- ◆ Necesito todos los departamentos excepto aquellos cuyo número NO empiece por 'D'

```
SELECT DEPTNO, DEPTNAME  
FROM DEPARTMENT  
WHERE DEPTNO NOT LIKE 'D%' ;
```

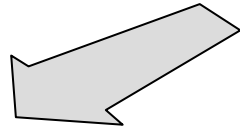


DEPTNO	DEPTNAME
A00	SPIFFY COMPUTER SERVICE DIV.
B01	PLANNING
C01	INFORMATION CENTER
E01	SUPPORT SERVICES
E11	OPERATIONS
E21	SOFTWARE SUPPORT

# Expresiones y renombramiento de columnas

---

```
SELECT EMPNO, SALARY, COMM,  
       SALARY+COMM AS INCOME  
FROM EMPLOYEE  
WHERE SALARY < 20000  
ORDER BY EMPNO
```



EMPNO	SALARY	COMM	INCOME
000210	18270.00	1462.00	19732.00
000250	19180.00	1534.00	20714.00
000260	17250.00	1380.00	18630.00
000290	15340.00	1227.00	16567.00
000300	17750.00	1420.00	19170.00
000310	15900.00	1272.00	17172.00
000320	19950.00	1596.00	21546.00



# Tipos SQL y valores literales

---

- ◆ La norma SQL define un conjunto de tipos para las columnas de las tablas.
  - Habitualmente cada SGBD tiene tipos propios o particularidades para los tipos de la norma SQL.
- ◆ Es necesario consultar el manual del SGBD para obtener información acerca de los tamaños máximos de almacenamiento.
  - En el caso de cadenas, cual es la longitud máxima de almacenamiento.
  - En el caso de tipos numéricos, cual es el rango de valores posibles.

# Tipos SQL y valores literales

---

Tipo	Ejemplo
BIGINT	8589934592
INTEGER	186282
SMALLINT	186
NUMERIC(8,2)	999999.99 (precisión, escala)
DECIMAL(8,2)	999999.99 (precisión, escala)
REAL	6.02257E23
DOUBLE PRECISION	3.141592653589
FLOAT	6.02257E23
CHARACTER( <i>max</i> )	'GREECE' (15 caracteres)
VARCHAR( <i>n</i> )	'hola'
DATE	date 'YYYY-MM-DD'
TIME	time 'hh:mm:ss.ccc'
TIMESTAMP	timestamp 'YYYY-MM-DD hh:mm:ss.ccc'

# Expresiones

---

- ◆ Aunque SQL no es un lenguaje de programación de propósito general, permite definir expresiones calculadas.
- ◆ Estas expresiones pueden contener
  - Referencias a columnas
  - Valores literales
  - Operadores aritméticos
  - Llamadas a funciones
- ◆ Los operadores aritméticos son los habituales: +, -, \* y /
  - Estos operadores sólo funcionan con valores numéricos.
  - Los operadores '+' y '-' habitualmente funcionan para fechas.
- ◆ Aunque las normas SQL definen un conjunto mínimo de funciones, los SGBD proporcionan una gran variedad.
  - Es necesario consultar el manual del SGBD particular.

# Funciones matemáticas comunes

---

Descripción	IBM DB2	SQL Server	Oracle	MySQL
Valor absoluto	ABSs	ABS	ABS	ABS
Menor entero $\geq$ valor	CEIL	CEILING	CEIL	CEILING
Menor entero $\leq$ valor	FLOOR	FLOOR	FLOOR	FLOOR
Potencia	POWER	POWER	POWER	POWER
Redondeo a un número de cifras decimales	ROUND	ROUND	ROUND	ROUND
Módulo	MOD.	%	MOD.	%

# Funciones de cadena

---

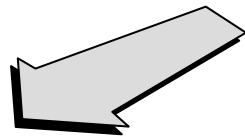
Descripción	IBM DB2	SQL Server	Oracle	MySQL
Convierte todos los caracteres a minúsculas	LOWER	LOWER	LOWER	LOWER
Convierte todos los caracteres a mayúsculas	UPPER	UPPER	UPPER	UPPER
Elimina los blancos del final de la cadena	RTRIM	RTRIM	RTRIM	RTRIM
Elimina los blancos del comienzo de la cadena	LTRIM	LTRIM	LTRIM	LTRIM
Devuelve una subcadena	SUBSTR	SUBSTRING	SUBSTR	SUBSTRING
Concatena dos cadenas	CONCAT	+	CONCAT	CONCAT

# Operaciones aritméticas

- ◆ Necesito obtener el salario, la comisión y los ingresos totales de todos los empleados que tengan un salario menor de 20000€ , clasificado por número de empleado

```
SELECT  EMPNO, SALARY, COMM,  
        SALARY + COMM
```

```
FROM EMPLOYEE  
WHERE SALARY < 20000  
ORDER BY EMPNO
```

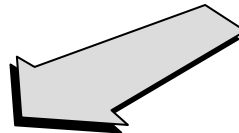


EMPNO	SALARY	COMM	
000210	18270.00	1462.00	19732.00
000250	19180.00	1534.00	20714.00
000260	17250.00	1380.00	18630.00
000290	15340.00	1227.00	16567.00
000300	17750.00	1420.00	19170.00
000310	15900.00	1272.00	17172.00
000320	19950.00	1596.00	21546.00

# Operaciones aritméticas (cont.)

---

```
SELECT  EMPNO, SALARY,  
        SALARY*1.0375  
FROM    EMPLOYEE  
WHERE   SALARY < 20000  
ORDER  BY EMPNO
```

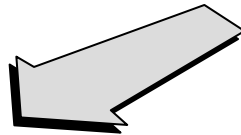


EMPNO	SALARY	
000210	18270.00	18955.125000
000250	19180.00	19899.250000
000260	17250.00	17896.875000
000290	15340.00	15915.250000
000300	17750.00	18415.625000
000310	15900.00	16496.250000
000320	19950.00	20698.125000

# Expresiones en predicados

---

```
SELECT  EMPNO, SALARY,  
        (COMM/SALARY)*100  
FROM    EMPLOYEE  
WHERE   (COMM/SALARY) * 100 > 8  
ORDER BY EMPNO
```



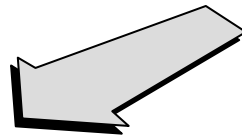
EMPNO	COMM	SALARY	
000140	2274.00	28420.00	8.001400
000210	1462.00	18270.00	8.002100
000240	2301.00	28760.00	8.000600
000330	2030.00	25370.00	8.001500



# Uso de funciones

---

```
SELECT  EMPNO, SALARY,  
        TRUNC (SALARY*1.0375, 2)  
FROM EMPLOYEE  
WHERE SALARY < 20000  
ORDER BY EMPNO
```

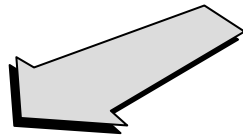


EMPNO	SALARY	
000210	18270.00	18955.12
000250	19180.00	19899.25
000260	17250.00	17896.87
000290	15340.00	15915.25
000300	17750.00	18415.62
000310	15900.00	16496.25
000320	19950.00	20698.12

## Uso de funciones (cont.)

---

```
SELECT LASTNAME & ', ' & FIRSTNAME ) AS NAME  
FROM EMPLOYEE  
WHERE WORKDEPT = 'A00'  
ORDER BY NAME
```



NAME

HAAS, CHRISTA  
LUCCHESI, VINCENZO  
O'CONNELL, SEAN

# Operadores de conjunto

---

- ◆ SQL incluye las operaciones:
  - **UNION**
  - **INTERSECT**
  - **EXCEPT** (**MINUS** en Oracle)
- ◆ Por definición los operadores de conjunto eliminan las tuplas duplicadas.
  - Para retener duplicados se debe utilizar **<Operador> ALL**

# UNION

---

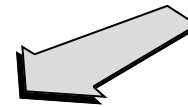
- ◆ Cada **SELECT** debe tener el mismo número de columnas
- ◆ Las columnas correspondientes deben tener tipos de datos compatibles
- ◆ **UNION** elimina duplicados
- ◆ Si se indica, el **ORDER BY** debe ser la última cláusula de la sentencia

# UNION

Cada entrada debe tener 2 lineas: la primera debe incluir el número y nombre del director y la segunda el número y el nombre del departamento.



```
SELECT      MGRNO , 'Dept.:', DEPTNAME
FROM        DEPARTMENT
UNION
SELECT      MGRNO, 'Mgr.:', LASTNAME
FROM        DEPARTMENT D, EMPLOYEE E
WHERE       D.MGRNO = E.EMPNO
ORDER BY    1,2 DESC
```



MGRNO		DEPTNAME
000010	Mgr.:	HAAS
000010	Dept.:	SPIFFY COMPUTER SERVICE DIV.
000020	Mgr.:	THOMPSON
000020	Dept.:	PLANNING
000030	Mgr.:	KWAN
000030	Dept.:	INFORMATION CENTER
000050	Mgr.:	GEYER
000050	Dept.:	SUPPORT SERVICES

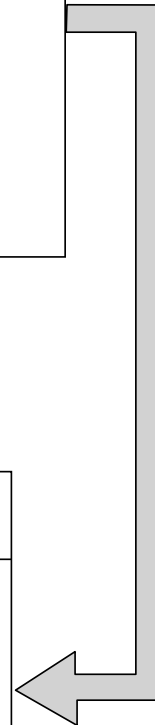
# Consultar más de una tabla

EMPLOYEE

EMPNO	LASTNAME	WORKDEPT...
000010	HAAS	A00
000020	THOMPSON	C01
000030	KWAN	C01
000040	PULASKI	D21
.	.	.

DEPARTMENT

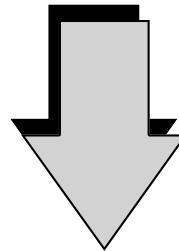
DEPTNO	DEPTNAME	...
A00	SPIFFY COMPUTER SERVICE DIV.	
C01	INFORMATION CENTER	
D01	DEVELOPMENT CENTER	
D21	ADMINISTRATION SYSTEMS	
.	.	.



# Sintaxis del JOIN: formato 1

---

```
SELECT  EMPNO, LASTNAME, WORKDEPT, DEPTNAME
FROM    EMPLOYEE,
        DEPARTMENT
WHERE   WORKDEPT = DEPTNO
        AND LASTNAME = 'HAAS'
```



EMPNO	LASTNAME	WORKDEPT	DEPTNAME
000010	HAAS	A00	SPIFFY COMPUTER SERVICE DIV.

# JOIN de tres tablas

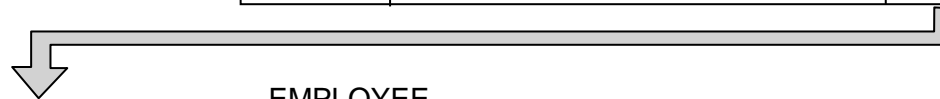
PROJECT

PROJNO	PROJNAME	DEPTNO	...
AD3100	ADMIN SERVICES	D01	
AD3110	GENERAL AD SYSTEMS	D21	
AD3111	PAYROLL PROGRAMMING	D21	
AD3112	PERSONELL PROGRAMMING	D21	
AD3113	ACCOUNT. PROGRAMMING	D21	
IF1000	QUERY SERVICES	C01	



DEPARTMENT

DEPTNO	DEPTNAME	MGRNO	...
A00	SPIFFY COMPUTER SERVICE DIV.	000010	
B01	PLANNING	000020	
C01	INFORMATION CENTER	000030	
D01	DEVELOPMENT CENTER	-----	
D11	MANUFACTURING SYSTEMS	000060	
D21	ADMINISTRATION SYSTEMS	000070	
E01	SUPPORT SERVICES	000050	



EMPLOYEE

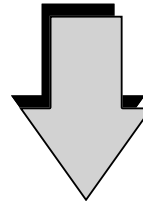
EMPNO	FIRSTNAME	MIDINIT	LASTNAME	...
000010	CHRISTA	I	HAAS	
000020	MICHAEL	L	THOMPSON	
000030	SALLY	A	KWAN	
000050	JOHN	B	GEYER	
000060	IRVING	F	STERN	
000070	EVA	D	PULASKI	
000090	EILEEN	W	HENDERSON	
000100	THEODORE	Q	SPENNER	



# JOIN de tres tablas

---

```
SELECT  PROJNO, PROJECT.DEPTNO, DEPTNAME, MGRNO, LASTNAME
FROM    PROJECT,
        DEPARTMENT,
        EMPLOYEE
WHERE   PROJECT.DEPTNO      = DEPARTMENT.DEPTNO
        AND DEPARTMENT.MGRNO = EMPLOYEE.EMPNO
        AND DEPARTMENT.DEPTNO = 'D21'
ORDER BY PROJNO
```

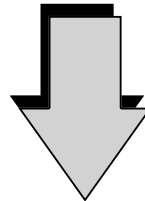


PROJNO	DEPTNO	DEPTNAME	MGRNO	LASTNAME
AD3110	D21	ADMINISTRATION SYSTEMS	000070	PULASKI
AD3111	D21	ADMINISTRATION SYSTEMS	000070	PULASKI
AD3112	D21	ADMINISTRATION SYSTEMS	000070	PULASKI
AD3113	D21	ADMINISTRATION SYSTEMS	000070	PULASKI

# Nombre de correlación (P, D, E)

---

```
SELECT  PROJNO, P.DEPTNO, DEPTNAME, MGRNO, LASTNAME
FROM    PROJECT P,
        DEPARTMENT D,
        EMPLOYEE E
WHERE   P.DEPTNO = D.DEPTNO
        AND D.MGRNO = E.EMPNO
        AND D.DEPTNO = 'D21'
ORDER BY PROJNO
```

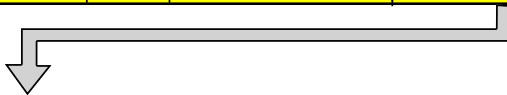


PROJNO	DEPTNO	DEPTNAME	MGRNO	LASTNAME
AD3110	D21	ADMINISTRATION SYSTEMS	000070	PULASKI
AD3111	D21	ADMINISTRATION SYSTEMS	000070	PULASKI
AD3112	D21	ADMINISTRATION SYSTEMS	000070	PULASKI
AD3113	D21	ADMINISTRATION SYSTEMS	000070	PULASKI

# JOIN de una tabla consigo misma

1. Recuperar la fila de un empleado de la tabla EMPLOYEE (E)

EMPNO	...	LASTNAME	WORKDEPT	...	BIRTHDATE	...
000100		SPENSER	E21		1956-12-18	
000330		LEE	E21		1941-07-18	



2. Recuperar el nº departamento de DEPARTMENT (D)

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
E21	SOFTWARE SUPPORT	000100	E21



3. Recuperar el director en EMPLOYEE (M)

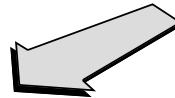
EMPNO	...	LASTNAME	WORKDEPT	...	BIRTHDATE	...
000100		SPENSER	E21		1956-12-18	
000330		LEE	E21		1941-07-18	

# JOIN de una tabla consigo misma

¿Qué empleados  
son mayores que  
su director?



```
SELECT  E.EMPNO, E.LASTNAME,  
        E.BIRTHDATE, M.BIRTHDATE, M.EMPNO  
FROM    EMPLOYEE E, EMPLOYEE M,  
        DEPARTMENT D  
WHERE   E.WORKDEPT = D.DEPTNO  
        AND      D.MGRNO   = M.EMPNO  
        AND      E.BIRTHDATE < M.BIRTHDATE
```



EMPNO	LASTNAME	BIRTHDATE	BIRTHDATE	EMPNO
000110	LUCCHESI	1929-11-05	1933-08-14	000010
000130	QUINTANA	1925-09-15	1941-05-11	000030
000200	BROWN	1941-05-29	1945-07-07	000060
000230	JEFFERSON	1935-05-30	1953-05-26	000070
000250	SMITH	1939-11-12	1953-05-26	000070
000260	JOHNSON	1936-10-05	1953-05-26	000070
000280	SCHNEIDER	1936-03-28	1941-05-15	000090
000300	SMITH	1936-10-27	1941-05-15	000090
000310	SETRIGHT	1931-04-21	1941-05-15	000090
000320	MEHTA	1932-08-11	1956-12-18	000100
000330	LEE	1941-07-18	1956-12-18	000100
000340	GOUNOT	1926-05-17	1956-12-18	000100

# Funciones de columna

---

- ◆ Las funciones de columna o *funciones de agregación* son funciones que toman una colección (conjunto o multiconjunto) de valores de entrada y devuelve un solo valor.
- ◆ Las funciones de columna disponibles son: **AVG**, **MIN**, **MAX**, **SUM**, **COUNT**.
- ◆ Los datos de entrada para **SUM** y **AVG** deben ser una colección de números, pero el resto de operadores pueden operar sobre colecciones de datos de tipo no numérico.

# Funciones de columna

---

- ◆ Por defecto las funciones se aplican a todas las tuplas resultantes de la consulta.
- ◆ Podemos agrupar las tuplas resultantes para poder aplicar las funciones de columna a grupos específicos utilizando la cláusula **GROUP BY**.
- ◆ En la cláusula **SELECT** de consultas que utilizan funciones de columna solamente pueden aparecer funciones de columna.
  - En caso de utilizar **GROUP BY**, también pueden aparecer columnas utilizadas en la agrupación.
- ◆ Adicionalmente se pueden aplicar condiciones sobre los grupos utilizando la cláusula **HAVING**.

# Funciones de columna

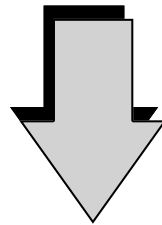
---

- ◆ Cálculo del total → **SUM (expresión)**
- ◆ Cálculo de la media → **AVG (expresión)**
- ◆ Obtener el valor mínimo → **MIN (expresión)**
- ◆ Obtener el valor máximo → **MAX (expresión)**
- ◆ Contar el número de filas que satisfacen la condición de búsqueda → **COUNT (\*)**
  - Los valores NULL SI se cuentan.
- ◆ Contar el número de valores distintos en una columna → **COUNT (DISTINCT nombre-columna)**
  - Los valores NULL NO se cuenta.

# Funciones de columna

---

```
SELECT    SUM(SALARY) AS SUM,  
          AVG(SALARY) AS AVG,  
          MIN(SALARY) AS MIN,  
          MAX(SALARY) AS MAX,  
          COUNT(*)      AS COUNT,  
          COUNT(DISTINCT WORKDEPT) AS DEPT  
FROM      EMPLOYEE
```



SUM	AVG	MIN	MAX	COUNT	DEPT
873715.00	27303.59375000	15340.00	52750.00	32	8



# GROUP BY

Necesito conocer los salarios de todos los empleados de los departamentos A00, B01, y C01. Además, para estos departamentos quiero conocer su masa salarial.



```
SELECT WORKDEPT, SALARY
FROM EMPLOYEE
WHERE WORKDEPT IN ('A00', 'B01', 'C01')
ORDER BY WORKDEPT
```



WORKDEPT	SALARY
A00	52750.00
A00	46500.00
A00	29250.00
B01	41250.00
C01	38250.00
C01	23800.00
C01	28420.00

```
SELECT WORKDEPT, SUM(SALARY) AS SUM
FROM EMPLOYEE
WHERE WORKDEPT IN ('A00', 'B01', 'C01')
GROUP BY WORKDEPT
ORDER BY WORKDEPT
```



WORKDEPT	SUM
A00	128500.00
B01	41250.00
C01	90470.00

Lenguaje SQL

# GROUP BY-HAVING

Ahora sólo quiero ver los departamentos  
cuya masa salarial sea superior a 50000



```
SELECT WORKDEPT, SUM(SALARY) AS SUM
FROM EMPLOYEE
WHERE WORKDEPT IN ('A00', 'B01', 'C01')
GROUP BY WORKDEPT
ORDER BY WORKDEPT
```



WORKDEPT	SUM
A00	128500.00
B01	41250.00
C01	90470.00

```
SELECT WORKDEPT, SUM(SALARY) AS SUM
FROM EMPLOYEE
WHERE WORKDEPT IN ('A00', 'B01', 'C01')
GROUP BY WORKDEPT
HAVING SUM(SALARY) > 50000
ORDER BY WORKDEPT
```



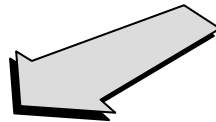
WORKDEPT	SUM
A00	128500.00
C01	90470.00

# GROUP BY-HAVING

Necesito, agrupado por departamento, los trabajadores que no sean managers, designer, y fieldrep, con una media de salario mayor que 25000€.



```
SELECT      WORKDEPT, JOB, AVG(SALARY) AS AVG
FROM        EMPLOYEE
WHERE       JOB NOT IN ('MANAGER', 'DESIGNER', 'FIELDREP')
GROUP BY    WORKDEPT, JOB
HAVING      AVG(SALARY) > 25000
ORDER BY    WORKDEPT, JOB
```



WORKDEPT	JOB	AVG
A00	CLERK	29250.00000000
A00	PRES	52750.00000000
A00	SALESREP	46500.00000000
C01	ANALYST	26110.00000000

# GROUP BY-HAVING

- Mostrar los departamentos con más de un empleado

SELECT 1

```
SELECT      WORKDEPT, COUNT(*) AS NUMB
FROM        EMPLOYEE
GROUP BY    WORKDEPT
ORDER BY    NUMB, WORKDEPT
```



WORKDEPT	NUMB
B01	1
E01	1
A00	3
C01	3
E21	4
E11	5
D21	6
D11	9

SELECT 2

```
SELECT      WORKDEPT, COUNT(*) AS NUMB
FROM        EMPLOYEE
GROUP BY    WORKDEPT
HAVING      COUNT(*) > 1
ORDER BY    NUMB, WORKDEPT
```



WORKDEPT	NUMB
A00	3
C01	3
E21	4
E11	5
D21	6
D11	9

# GROUP BY-HAVING

SELECT 1

```
SELECT      WORKDEPT, AVG(EDLEVEL) AS ED,  
            AVG(YEAR(CURRENT_DATE-HIREDATE))  
            AS YEARS  
FROM        EMPLOYEE  
GROUP BY   WORKDEPT  
ORDER BY   2
```



WORKDEPT	ED	YEARS
E11	14	27
E21	15	31
D21	15	22
E01	16	49
D11	16	24
A00	17	35
B01	18	24
C01	18	23

SELECT 2

```
SELECT      WORKDEPT, AVG(EDLEVEL) AS ED,  
            AVG(YEAR(CURRENT_DATE-HIREDATE))  
            AS YEARS  
FROM        EMPLOYEE  
GROUP BY   WORKDEPT  
HAVING     AVG(YEAR(CURRENT_DATE-HIREDATE)) >= 30  
ORDER BY   2
```



WORKDEPT	ED	YEARS
E21	15	31
E01	16	49
A00	17	35

# GROUP BY-HAVING

SELECT 1

```
SELECT      WORKDEPT, AVG(EDLEVEL) AS ED,  
            MIN(BONUS) AS MIN  
FROM        EMPLOYEE  
GROUP BY    WORKDEPT
```



WORKDEPT	ED	MIN
A00	17	600.00
B01	18	800.00
C01	18	500.00
D11	16	400.00
D21	15	300.00
E01	16	800.00
E11	14	300.00

SELECT 2

```
SELECT      WORKDEPT, AVG(EDLEVEL) AS ED,  
            MIN(BONUS) AS MIN  
FROM        EMPLOYEE  
GROUP BY    WORKDEPT  
HAVING      MIN(BONUS) = 300  
ORDER BY    2
```



WORKDEPT	ED	MIN
E11	14	300.00
D21	15	300.00

# Ejecución de consultas SELECT

---

- ◆ El orden de ejecución de una consulta es el siguiente:
  1. Se aplica el predicado **WHERE** a las tuplas del producto cartesiano/join/vista que hay en el **FROM**.
  2. Las tuplas que satisfacen el predicado de **WHERE** son colocadas en grupos siguiendo el patrón **GROUP BY**.
  3. Se ejecutan la cláusula **HAVING** para cada grupo de tuplas anterior.
  4. Los grupos obtenidos tras aplicar **HAVING** son los que serán procesados por **SELECT**, que calculará, en los casos que se incluyan, las funciones de agregación que le acompañan.
  5. A las tuplas resultantes de los pasos anteriores se le aplica la ordenación descrita en la cláusula **ORDER BY**.

# Subconsulta con IN



Tabla DEPARTMENT

<u>DEPTNO</u>	<u>DEPTNAME</u>
A00	SPIFFY COMPUTER SERVICE
B01	PLANNING
C01	INFORMATION CENTER
...	...

```
SELECT  DEPTNO, DEPTNAME
FROM    DEPARTMENT
WHERE   DEPTNO NOT IN (SELECT DEPTNO
                       FROM PROJECT)
```

Resultado final

DEPTNO	DEPTNAME
A00	SPIFFY COMPUTER SERVICE

Resultado subconsulta

B01
C01
D01
D11
D21
E01
E11
E21



# Modificación de la BBDD

---

- ◆ Las instrucciones SQL que permiten modificar el estado de la BBDD son:
  - **INSERT** → Añade filas a una tabla
  - **UPDATE** → Actualiza filas de una tabla
  - **DELETE** → Elimina filas de una tabla

# La instrucción **INSERT**

---

## ◆ La **inserción** de tuplas se realiza con la sentencia **INSERT**,

- Es posible insertar directamente valores.
- O bien insertar el conjunto de resultados de una consulta.
- En cualquier caso, los valores que se insertan deben pertenecer al dominio de cada uno de los atributos de la relación.

## ◆ *Ejemplos:* CLIENTES (DNI, NOMBRE, DIR)

### ◆ La inserción

- **INSERT INTO** CLIENTES **VALUES** (1111, 'Mario', 'C/. Mayor, 3');

### ◆ Es equivalente a las siguientes sentencias

- **INSERT INTO** CLIENTES (NOMBRE, DIR, DNI) **VALUES** ('Mario', 'C/. Mayor, 3', 1111);
- **INSERT INTO** CLIENTES (DNI, DIR, NOMBRE) **VALUES** (1111, 'C/. Mayor, 3', 'Mario');

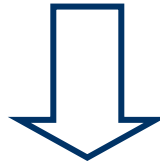
# Añadir una fila

---

```
INSERT INTO TESTEMP  
VALUES ('000111', 'SMITH', 'C01', '1998-06-25', 25000, NULL)
```



```
INSERT INTO TESTEMP(EMPNO, LASTNAME, WORKDEPT, HIREDATE, SALARY)  
VALUES ('000111', 'SMITH', 'C01', '1998-06-25', 25000)
```



<u>EMPNO</u>	<u>LASTNAME</u>	<u>WORKDEPT</u>	<u>HIREDATE</u>	<u>SALARY</u>	<u>BONUS</u>
000111	SMITH	C01	1998-06-25	25000.00	-

# Añadir varias filas

---

- ◆ *Ejemplo:*

- ◆ Para la siguiente base de datos, queremos incluir en la relación GRUPOS a todos los grupos, junto con su número de álbumes publicados:

- GRUPOS (NOMBRE, ALBUMES)    LP (TIT, GRUPO, ANIO, NUM\_CANC)

- ◆ Solución:

```
INSERT INTO GRUPOS
  SELECT GRUPO, COUNT (DISTINCT TIT) FROM LP
  GROUP BY GRUPO;
```

- ◆ En SQL se **prohíbe** que la **consulta** que se incluye en una **cláusula INSERT** haga referencia a la misma tabla en la que se quieren insertar las tuplas.
  - En ORACLE sí está permitido

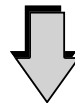
# Añadir varias filas (cont.)

TESTEMP

EMPNO	LASTNAME	WORKDEPT	HIREDATE	SALARY	BONUS
-------	----------	----------	----------	--------	-------

INSERT INTO TESTEMP

SELECT EMPNO, LASTNAME, WORKDEPT, HIREDATE, SALARY, BONUS  
FROM EMPLOYEE  
WHERE EMPNO <= '000050'



<u>EMPNO</u>	<u>LASTNAME</u>	<u>WORKDEPT</u>	<u>HIREDATE</u>	<u>SALARY</u>	<u>BONUS</u>
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00
000111	SMITH	C01	1998-06-25	25000.00	- - - - -

# La instrucción **UPDATE**

---

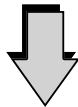
- ◆ La **modificación** de tuplas se realiza con la sentencia **UPDATE**,
  - Es posible elegir el conjunto de tuplas que se van a actualizar usando la clausula **WHERE**.
- ◆ *Ejemplos:* CUENTAS (COD, DNI, NSUCURS, SALDO)
- ◆ Suma del 5% de interés a los saldos de todas las cuentas.
  - **UPDATE** CUENTAS **SET** SALDO = SALDO \* 1.05;
- ◆ Suma del 1% de bonificación a aquellas cuentas cuyo saldo sea superior a 100.000 €.
  - **UPDATE** CUENTAS **SET** SALDO = SALDO \* 1.01  
**WHERE** SALDO > 100000;
- ◆ Modificación de DNI y saldo simultáneamente para el código 898.
  - **UPDATE** CUENTAS **SET** DNI='555', SALDO=10000  
**WHERE** COD **LIKE** '898';

# Modificar datos

Antes:

<u>EMPNO</u>	<u>LASTNAME</u>	<u>WORKDEPT</u>	<u>HIREDATE</u>	<u>SALARY</u>	<u>BONUS</u>
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00
000111	SMITH	C01	1998-06-25	25000.00	- - - - -

UPDATE TESTEMP  
SET SALARY = SALARY + 1000  
WHERE WORKDEPT = 'C01'



Después:

<u>EMPNO</u>	<u>LASTNAME</u>	<u>WORKDEPT</u>	<u>HIREDATE</u>	<u>SALARY</u>	<u>BONUS</u>
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	39250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00
000111	SMITH	C01	1998-06-25	26000.00	- - - - -

# La instrucción **DELETE**

---

- ◆ La **eliminación** de tuplas se realiza con la sentencia **DELETE**:
  - **DELETE FROM R WHERE P;** -- WHERE es opcional
  - Elimina tuplas completas, no columnas. Puede incluir subconsultas.
- ◆ *Ejemplos:* para la BD de CLIENTES, CUENTAS, SUCURSALES.
- ◆ Eliminar todas cuentas con código entre 1000 y 1100.
  - **DELETE FROM CUENTAS WHERE COD BETWEEN 1000 AND 1100;**
- ◆ Eliminar todas las cuentas del cliente “Jose María García”.
  - **DELETE FROM CUENTAS WHERE DNI IN (SELECT DNI FROM CLIENTES WHERE NOMBRE LIKE 'Jose María García');**
- ◆ Eliminar todas las cuentas de sucursales situadas en "Chinchón".
  - **DELETE FROM CUENTAS WHERE NSUCURS IN (SELECT NSUC FROM SUCURSALES WHERE CIUDAD LIKE 'Chinchón');**

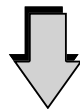


# Borrar filas

Antes:

<u>EMPNO</u>	<u>LASTNAME</u>	<u>WORKDEPT</u>	<u>HIREDATE</u>	<u>SALARY</u>	<u>BONUS</u>
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00
000111	SMITH	C01	1998-06-25	25000.00	- - - - -

DELETE FROM TESTEMP  
WHERE EMPNO = '000111'



Después:

<u>EMPNO</u>	<u>LASTNAME</u>	<u>WORKDEPT</u>	<u>HIREDATE</u>	<u>SALARY</u>	<u>BONUS</u>
000010	HAAS	A00	1965-01-01	52750.00	1000.00
000020	THOMPSON	B01	1973-10-10	41250.00	800.00
000030	KWAN	C01	1975-04-05	38250.00	800.00
000050	GEYER	E01	1949-08-17	40175.00	800.00

# Transacciones

---

- ◆ Una transacción es una operación lógica o un conjunto de órdenes, cuya ejecución da forma a una unidad de trabajo.
  - ◆ Cada transacción debe poseer las propiedades denominadas ACID (**acrónimo para los conceptos *Atomicity*, *Consistency*, *Isolation* y *Durability*** )
- 
- ◆ A: Atomicidad
  - ◆ C: Consistencia
  - ◆ I : Aislamiento
  - ◆ D: Durabilidad

# Transacciones

---

- ◆ **A: Atomicidad:** se refiere a la atomicidad de las transacciones, lo que significa que el sistema permite que se lleven a cabo las operaciones atómicas. **Esta propiedad indica que, para que una transacción se dé por terminada, deben haberse realizado todas sus partes o ninguna de ellas.**
- ◆ **C: Consistencia:** la capacidad que tiene un sistema para iniciar solo operaciones que puede concluir. **Una transacción debe llevar al sistema de un estado válido a otro que también lo sea.**

# Transacciones

---

- ◆ I : Aislamiento: la realización de una operación no debería afectar a las otras, debido a que **cada una de las transacciones debe ejecutarse en aislamiento total**, sin importar si se llevan a cabo de manera simultánea. El estado de una transacción no debe ser visible por otra.
- ◆ D: Durabilidad: si se ha realizado una determinada operación (*commit*), esta tiene la **capacidad de persistir y no puede deshacerse incluso si el sistema falla** o se presentan eventos como errores, caídas o pérdida de alimentación eléctrica, entre otros. Esto **implica que los datos y cambios en una transacción que ya se han realizado deben ser permanentes** y no puede ocurrir una pérdida de los mismos en el sistema.

# Transacciones

---

- ◆ BEGIN TRAN

- ◆ sentencia 1

- ◆ ...

- ◆ sentencia n

- ◆ COMMIT TRAN

- ◆ START TRAN

- ◆ sentencia 1

- ◆ ...

- ◆ sentencia n

- ◆ COMMIT