

# INTEGRIDAD DE DATOS

---

M. EN C. NIELS HENRIK NAVARRETE MANZANILLA



# CLASIFICACIÓN

---

- **Integridad en entidades**

No duplicar datos en la tabla

- **Dominio de integridad**

Determina que entradas son validas de acuerdo al tipo de dato, el formato o el rango de valores

- **Referencia de integridad**

Los registros no pueden ser borrados cuando sean utilizados por otros registros

- **Integridad definida por el usuario**

Aplica unas reglas de negocio que no están en las entidades, de dominio o de integridad referencial.

# NORMALIZACIÓN DE BASE DE DATOS.

---

Es el proceso de organizar eficientemente los datos y la razón es:

- Eliminar la redundancia de los datos, es decir, el almacenamiento de los mismos datos en una o varias tablas.
- Asegurar que las dependencias tengan sentido.

Estos puntos permiten reducir el espacio de almacenamiento y asegura que los datos sean almacenados lógicamente.

Existe un proceso que consisten en una serie de directrices que ayudan a guiar en la creación de la estructura de la base de datos.



# NORMALIZACIÓN DE BASE DE DATOS.

---

El objetivo de normalizar es organizar los datos, cumpliendo las siguientes reglas:

- Primera forma normal (1NF)
- Segunda forma normal (2NF)
- Tercera forma normal (3NF)

# PRIMERA FORMA NORMAL

---

Establece las reglas fundamentales para la organización de la base de datos.

- Define los datos requeridos y necesarios para ser almacenados en las tablas.
- Asegurar que no existen conjuntos de datos repetidos en otras tablas.
- Asegurar de que exista al menos una llave primaria en cada tabla creada.



# PRIMERA FORMA NORMAL

- 
- Primera regla:

Determina como deben de almacenarse los datos y organizarse dentro de las columnas ( el tipo de dato de cada una de las entradas).

Ej.:

Se crea una tabla con las ubicaciones de ciertas reuniones de trabajo y la tabla se llama: Miembros de detalle.



# PRIMERA FORMA NORMAL

- Segunda regla:

El siguiente paso es asegurarse que no existan conjunto de datos repetidos.

```
CREATE TABLE CUSTOMERS (  
    ID INT NOT NULL,  
    NAME VARCHAR (20) NOT NULL,  
    AGE INT NOT NULL,  
    ADDRESS CHAR (25),  
    ORDERS VARCHAR (155)  
);
```

ID	NAME	AGE	ADDRESS	ORDERS
100	Sachin	36	Lower West Side	Cannon XL-200
100	Sachin	36	Lower West Side	Battery XL-200
100	Sachin	36	Lower West Side	Tripod Large

- Tercera regla:

Cada tabla creada debe contener una llave primaria.

# PRIMERA FORMA NORMAL

```
CREATE TABLE CUSTOMERS (  
    ID INT NOT NULL,  
    NAME VARCHAR (20) NOT NULL,  
    AGE INT NOT NULL,  
    ADDRESS CHAR (25),  
    PRIMARY KEY (ID)  
);
```

```
CREATE TABLE ORDERS (  
    ID INT NOT NULL,  
    CUSTOMER_ID INT NOT NULL,  
    ORDERS VARCHAR(155),  
    PRIMARY KEY (ID)  
);
```

ID	NAME	AGE	ADDRESS	ORDERS
100	Sachin	36	Lower West Side	Cannon XL-200
100	Sachin	36	Lower West Side	Battery XL-200
100	Sachin	36	Lower West Side	Tripod Large

ID	NAME	AGE	ADDRESS
100	Sachin	36	Lower West Side

ID	CUSTOMER ID	ORDERS
10	100	Cannon XL-200
11	100	Battery XL-200
12	100	Tripod Large



# SEGUNDA FORMA NORMAL

---

Es necesario haber establecido antes los criterios de la primera forma normal, para posteriormente verificar que no deben existir dependencias parciales en ninguna de las columnas de las llaves primarias.

Ej.

Consideremos la relación customer-order y deseamos almacenar los siguientes datos:

```
CREATE TABLE CUSTOMERS (  
    CUST_ID      INT                NOT NULL,  
    CUST_NAME    VARCHAR (20)       NOT NULL,  
    ORDER_ID     INT                NOT NULL,  
    ORDER_DETAIL VARCHAR (20)       NOT NULL,  
    SALE_DATE    DATETIME,  
    PRIMARY KEY  (CUST_ID, ORDER_ID)  
);
```

# SEGUNDA FORMA NORMAL

- 
- Las llaves primarias son (CUST\_ID , ORDER\_ID), como deben ser únicas, se asume que muy difícilmente el cliente (customer) ordene el mismo producto (comida, etc.).

De cualquier manera, no se cumple la segunda forma normal por que existen dependencias parciales en las llaves primarias y en las columnas.

CUST\_NAME depende de CUST\_ID, pero estas no están ligadas entre el CUSTOMER\_NAME y lo que se compro.

ORDER\_DETAIL y SALE\_DATE dependen de ORDER\_ID, pero ellos no dependen de CUST\_ID, porque no hay una manera de relacionar ORDER\_DETAIL o SALE\_DATE

# SEGUNDA FORMA NORMAL

- Para que se cumpla la segunda forma normal, es necesario separa las columnas en tres mas.
- Primero almacenaremos los datos de los clientes (customers) de la siguiente manera;

```
CREATE TABLE CUSTOMERS (  
    CUST_ID      INT                NOT NULL,  
    CUST_NAME    VARCHAR (20)       NOT NULL,  
    PRIMARY KEY (CUST_ID)  
);
```

```
CREATE TABLE ORDERS (  
    ORDER_ID     INT                NOT NULL,  
    ORDER_DETAIL VARCHAR (20)       NOT NULL,  
    PRIMARY KEY (ORDER_ID)  
);
```

```
CREATE TABLE CUSTMERORDERS (  
    CUST_ID      INT                NOT NULL,  
    ORDER_ID     INT                NOT NULL,  
    SALE_DATE    DATETIME,  
    PRIMARY KEY (CUST_ID, ORDER_ID)  
);
```

# TERCERA FORMA NORMAL

Cuando una tabla (relación) se encuentra en la tercera forma normal es cuando cumple los siguientes criterios:

- Esta en segunda forma normal
- Todos los campos (no llaves primarias) dependen de la llave primaria

En la tabla de clientes. El campo calle (Street), ciudad (city) y estado (state) están ligados al código postal (zip)

```
CREATE TABLE CUSTOMERS(  
    CUST_ID          INT          NOT NULL,  
    CUST_NAME        VARCHAR (20)  NOT NULL,  
    DOB              DATE,  
    STREET            VARCHAR(200),
```

```
    CITY              VARCHAR(100) ,  
    STATE             VARCHAR(100) ,  
    ZIP               VARCHAR(12) ,  
    EMAIL_ID          VARCHAR(256) ,  
    PRIMARY KEY (CUST_ID)  
);
```

# TERCERA FORMA NORMAL

La dependencia del código postal con la dirección (address) se le llama dependencia transitiva. Para poder cumplir la tercera forma normal, es necesario mover la dirección (calle, ciudad, estado, etc.) en una tabla nueva, donde se pueda llamar su código postal.

```
CREATE TABLE ADDRESS (  
    ZIP          VARCHAR(12),  
    STREET       VARCHAR(200),  
    CITY         VARCHAR(100),  
    STATE        VARCHAR(100),  
    PRIMARY KEY (ZIP)  
);
```

```
CREATE TABLE CUSTOMERS (  
    CUST_ID      INT          NOT NULL,  
    CUST_NAME    VARCHAR (20)  NOT NULL,  
    DOB          DATE,  
    ZIP          VARCHAR(12),  
    EMAIL_ID     VARCHAR(256),  
    PRIMARY KEY (CUST_ID)  
);
```



# ¿QUÉ ES DEPENDENCIA TRANSITIVA?

- 
- Si hay un conjunto de atributos  $Z$  que no es un subconjunto de alguna clave de  $R$ , donde se mantiene  $X \rightarrow Z$  y  $Z \rightarrow Y$ .
  - Es cuando cualquier atributo en una tabla es dependiente de otro campo y éste es quien depende de la clave primaria.

# RESUMEN DE LAS FORMAS NORMALES

Forma Normal	Prueba	Remedio (normalización)
Primera (1FN)	La relación no debe tener atributos multi-valor o relaciones anidadas	Generar nuevas relaciones para cada atributo multi-valor o relación anidada
Segunda (2FN)	Para relaciones en las que la clave principal contiene varios atributos, un atributo no clave debe ser funcionalmente dependiente en una parte de clave principal	Descomponer y configurar una nueva relación por cada clave parcial con su(s) atributo(s) dependiente(s). Asegurarse de mantener una relación con la clave principal original y cualquier atributo que sea completa y funcionalmente dependiente de ella.
Terca (3FN)	La relación no debe tener un atributo no clave que este funcionalmente determinado por otro atributo no clave (o por un conjunto de atributos no clave). Esto es, debe ser una dependencia transitiva de un atributo no clave de la clave principal	Descomponer y configurar una relación que incluya el(los) atributo(s) no clave que determine(n) funcionalmente otro(s) atributo(s) no clave.

# REPASO Y EJEMPLOS DE FORMAS NORMALES

---

- Primera forma normal:

Requiere que no existan atributos multi-valores, así como tampoco grupos de repetición.

Un atributo multi-valor contiene más de un valor por ese campo en cada fila

Nro de Registro	Curso
12345	3100,3600,3900
54321	1300,2300,1200

Campo multi-valor

Nro de Registro	Curso1	Curso2	Curso3
12345	3100	3600	3900
54321	1300	2300	1200

Grupo repetidos

# REPASO Y EJEMPLOS DE FORMAS NORMALES

Nro de Registro	Curso
12345	3100,3600,3900
54321	1300,2300,1200

Campo multi-valor

Nro de Registro	Curso1	Curso2	Curso3
12345	3100	3600	3900
54321	1300	2300	1200

Grupo repetidos

Nro de Registro	Curso
12345	3100
12345	3600
12345	3900
54321	1300
54321	2300
54321	1200

Ordeno los registros, para agrupar los datos en común.

(Todavía No esta aún en 1NF)

# SQL SINTAXIS

---

EL CONJUNTO DE REGLAS SE LE CONOCE COMO SINTAXIS



# INTRODUCCIÓN

- 
- Todas las sentencias de SQL terminan con (;).
  - Es importante destacar que SQL es **case sensitive** (distinción de mayúsculas o minúsculas), lo cual significa SELECT y select tienen el mismo significado.
  - SQL si hace distinción entre el nombre de las tablas.
  - Las tablas deben en la base de datos.

# SENTENCIAS

## SQL SELECT Statement:

```
SELECT column1, column2....columnN  
FROM   table_name;
```

## SQL DISTINCT Clause:

```
SELECT DISTINCT column1, column2....columnN  
FROM   table_name;
```

## SQL COUNT Clause:

```
SELECT COUNT(column_name)  
FROM   table_name  
WHERE  CONDITION;
```

## SQL WHERE Clause:

```
SELECT column1, column2....columnN  
FROM   table name  
WHERE  CONDITION;
```

## SQL AND/OR Clause:

```
SELECT column1, column2....columnN  
FROM   table_name  
WHERE  CONDITION-1 {AND|OR} CONDITION-2;
```

## SQL IN Clause:

```
SELECT column1, column2....columnN  
FROM   table_name  
WHERE  column_name IN (val-1, val-2,...val-N);
```

# SENTENCIAS

## SQL BETWEEN Clause:

```
SELECT column1, column2....columnN
FROM   table name
WHERE  column_name BETWEEN val-1 AND val-2;
```

## SQL LIKE Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name LIKE { PATTERN };
```

## SQL WHERE Clause:

```
SELECT column1, column2....columnN
FROM   table name
WHERE  CONDITION;
```

## SQL GROUP BY Clause:

```
SELECT SUM(column_name)
FROM   table name
WHERE  CONDITION
GROUP BY column_name;
```

## SQL HAVING Clause:

```
SELECT SUM(column_name)
FROM   table name
WHERE  CONDITION
GROUP BY column name
HAVING (arithmetic function condition);
```

## SQL ORDER BY Clause:

```
SELECT column1, column2....columnN
FROM   table name
WHERE  CONDITION
ORDER BY column_name {ASC|DESC};
```

# SENTENCIAS

---

## SQL INSERT INTO Statement:

```
INSERT INTO table name( column1, column2....columnN)  
VALUES ( value1, value2....valueN);
```

## SQL UPDATE Statement:

```
UPDATE table_name  
SET column1 = value1, column2 = value2....columnN=valueN  
[ WHERE CONDITION ];
```

## SQL DELETE Statement:


```
DELETE FROM table name  
WHERE {CONDITION};
```

## SQL COMMIT Statement:

```
COMMIT;
```

## SQL ROLLBACK Statement:

```
ROLLBACK;
```





# SENTENCIAS

---

## SQL CREATE DATABASE Statement:

```
CREATE DATABASE database_name;
```

## SQL USE Statement:

```
USE DATABASE database_name;
```

## SQL DROP DATABASE Statement:

```
DROP DATABASE database_name;
```



# SQL TIPOS DE DATOS

---

ES UNA CARACTERÍSTICA QUE ESPECIFICA EL TIPO DE DATO QUE VA  
ALMACENAR LA COLUMNA



# TIPOS DE DATOS

- 
- Los tipos de datos se usan cuando se esta creando la tabla. Por lo que se debe elegir adecuadamente el tipo de dato y esto se conoce desde que se establece el requerimiento.
  - Por lo que SQL Server, ofrece seis categorías de tipo de datos para su uso.
    1. Numerico (1,2,3, 4.5)
    2. Fecha (09/09/2016)
    3. Cadenas (“Pruebas”)
    4. Caracteres Unicode (Cadenas amplias) (€¥¥)
    5. Binario (01010)
    6. Misc

# TIPOS DE DATOS (NUMÉRICOS)

## Exact Numeric Data Types:

DATA TYPE	FROM	TO
Bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	-2,147,483,648	2,147,483,647
Smallint	-32,768	32,767
Tinyint	0	255
Bit	0	1
Decimal	$-10^{38} + 1$	$10^{38} - 1$
Numeric	$-10^{38} + 1$	$10^{38} - 1$
Money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
Smallmoney	-214,748.3648	+214,748.3647

# TIPOS DE DATOS (NUMÉRICOS)

---

## Approximate Numeric Data Types:

DATA TYPE	FROM	TO
Float	-1.79E + 308	1.79E + 308
Real	-3.40E + 38	3.40E + 38

# TIPOS DE DATOS (FECHA)

---

## Date and Time Data Types:

DATA TYPE	FROM	TO
Datetime	Jan 1, 1753	Dec 31, 9999
Smalldatetime	Jan 1, 1900	Jun 6, 2079
Date	Stores a date like June 30, 1991	
Time	Stores a time of day like 12:30 P.M.	



# TIPOS DE DATOS (CADENA)

## Character Strings Data Types:

DATA TYPE	FROM	TO
Char	Char	Maximum length of 8,000 characters.( Fixed length non-Unicode characters)
Varchar	Varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).
varchar(max)	varchar(max)	Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
Text	text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

# TIPOS DE DATOS (CADENA)

---

## Unicode Character Strings Data Types:

DATA TYPE	Description
Nchar	Maximum length of 4,000 characters.( Fixed length Unicode)
Nvarchar	Maximum length of 4,000 characters.(Variable length Unicode)
nvarchar(max)	Maximum length of 231characters (SQL Server 2005 only).( Variable length Unicode)
Ntext	Maximum length of 1,073,741,823 characters. ( Variable length Unicode )

# TIPOS DE DATOS (BINARIO)

---

DATA TYPE	Description
Binary	Maximum length of 8,000 bytes(Fixed-length binary data )
Varbinary	Maximum length of 8,000 bytes.(Variable length binary data)
varbinary(max)	Maximum length of 231 bytes (SQL Server 2005 only). ( Variable length Binary data)
Image	Maximum length of 2,147,483,647 bytes. ( Variable length Binary Data)

# TIPOS DE DATOS (MISC)

---

DATA TYPE	Description
sql_variant	Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
timestamp	Stores a database-wide unique number that gets updated every time a row gets updated
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).
cursor	Reference to a cursor object
table	Stores a result set for later processing



# SQL OPERADORES

---

ES UNA PALABRA RESERVADA O UN CARÁCTER USADO PARA REALIZAR OPERACIONES (COMPARACIÓN Y ARITMETICOS)





# INTRODUCCIÓN

---

Los operadores son usados para especificar condiciones en una sentencia SQL y realizar múltiples operaciones tales como:

- Operadores aritméticos
- Operadores de comparación
- Operadores lógicos
- Operadores usados para realizar negaciones

# OPERADORES ARITMETICOS

---

OPERADOR	DESCRIPCIÓN	EJEMPLO
+	SUMA	$A+B$
-	RESTA	$A-B$
*	MULTIPLICACIÓN	$A*B$
/	DIVISIÓN	$A/B$
%	MODULO	$A\%B$

# OPERADORES ARITMETICOS

---

```
SQL> select 10+ 20;
+-----+
| 10+ 20 |
+-----+
|      30 |
+-----+
1 row in set (0.00 sec)
```

```
SQL> select 10 * 20;
+-----+
| 10 * 20 |
+-----+
|      200 |
+-----+
1 row in set (0.00 sec)
```

```
SQL> select 10 / 5;
+-----+
| 10 / 5 |
+-----+
| 2.0000 |
+-----+
1 row in set (0.03 sec)
```

```
SQL> select 12 % 5;
+-----+
| 12 % 5 |
+-----+
|        2 |
+-----+
1 row in set (0.00 sec)
```

# OPERADORES ARITMETICOS

---

OPERADOR	DESCRIPCIÓN	EJEMPLO
+	SUMA	A+B
-	RESTA	A-B
*	MULTIPLICACIÓN	A*B
/	DIVISIÓN	A/B
%	MODULO	A%B

# OPERADORES DE COMPARACIÓN

OPERADOR	DESCRIPCIÓN	EJEMPLO
=	Determina si dos valores son iguales. Si la condición se cumple regresa true	A=B
!=	Determina si dos valores son diferentes. Si son diferentes regresa true	A!=B
< >	Determina si dos valores son diferentes. Si son diferentes regresa true	A<>B
>	Determina si el valor de A es mayor que B. Si lo es regresa true	A>B
<	Determina si el valor de A es menor que B. Si lo es regresa true	A<B



# OPERADORES DE COMPARACIÓN

OPERADOR	DESCRIPCIÓN	EJEMPLO
>=	Determina si el valor de A es mayor o igual que B. Si lo es regresa true	A=B
<=	Determina si el valor de A es menor o igual que B. Si lo es regresa true	A!=B
!<	Determina si el valor de A no sea menor que B. Si lo es regresa true	A<>B
!>	Determina si el valor de A no sea mayor que B. Si lo es regresa true	A>B

# OPERADORES DE COMPARACIÓN

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY > 5000;
```

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

```
3 rows in set (0.00 sec)
```

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 2000;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	kaushik	23	Kota	2000.00

```
2 rows in set (0.00 sec)
```

# OPERADORES LÓGICOS

OPERADOR	DESCRIPCIÓN
ALL	Permite compara un valor con todos los valores de una lista
AND	Permite establecer condiciones de búsqueda (Where) A y B y C
ANY	Permite comparar un valor con cualquier valor que exista en la lista de valores
BETWEEN	Es usado para buscar valores dentro de un limite inferior a un limite superior.
EXISTS	Se utiliza para buscar un valor en una cierta columna de acuerdo algún criterio.
IN	Es usado para comprar un valor dentro de un conjunto de valores establecidos
LIKE	Es usado para comparar un valor con valores similares
NOT	Es una operador de negación (NOT EXISTS, NOT IN, etc)
OR	Permite establecer condiciones de búsqueda (Where) A o B o C
IS NULL	Comparar si un valor es NULL
UNIQUE	Busca en cada registro si son valores únicos (no duplicados)

```
SQL> SELECT * FROM CUSTOMERS WHERE AGE >= 25 AND SALARY >= 6500;
+----+-----+-----+-----+-----+
| ID | NAME      | AGE | ADDRESS | SALARY |
+----+-----+-----+-----+-----+
| 4  | Chaitali  | 25  | Mumbai  | 6500.00 |
| 5  | Hardik    | 27  | Bhopal   | 8500.00 |
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
SQL> SELECT * FROM CUSTOMERS WHERE AGE >= 25 OR SALARY >= 6500;
+----+-----+-----+-----+-----+
| ID | NAME      | AGE | ADDRESS | SALARY |
+----+-----+-----+-----+-----+
| 1  | Ramesh    | 32  | Ahmedabad | 2000.00 |
| 2  | Khilan    | 25  | Delhi    | 1500.00 |
| 4  | Chaitali  | 25  | Mumbai   | 6500.00 |
| 5  | Hardik    | 27  | Bhopal   | 8500.00 |
| 7  | Muffy     | 24  | Indore   | 10000.00 |
+----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
SQL> SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;
+----+-----+-----+-----+-----+
| ID | NAME      | AGE | ADDRESS | SALARY |
+----+-----+-----+-----+-----+
| 1  | Ramesh    | 32  | Ahmedabad | 2000.00 |
| 2  | Khilan    | 25  | Delhi    | 1500.00 |
| 3  | kaushik   | 23  | Kota     | 2000.00 |
| 4  | Chaitali  | 25  | Mumbai   | 6500.00 |
| 5  | Hardik    | 27  | Bhopal   | 8500.00 |
| 6  | Komal     | 22  | MP       | 4500.00 |
| 7  | Muffy     | 24  | Indore   | 10000.00 |
+----+-----+-----+-----+-----+
```

---

```
SQL> SELECT (15 + 6) AS ADDITION
+-----+
| ADDITION |
+-----+
|        21 |
+-----+
1 row in set (0.00 sec)
```

```
SQL> SELECT CURRENT_TIMESTAMP;
+-----+
| Current_Timestamp |
+-----+
| 2009-11-12 06:40:23 |
+-----+
1 row in set (0.00 sec)
```

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;
+-----+
| RECORDS |
+-----+
|        7 |
+-----+
1 row in set (0.00 sec)
```



# INSERT

```
INSERT INTO TABLE_NAME (column1, column2, column3, ...columnN) ]  
VALUES (value1, value2, value3, ...valueN);
```

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
```

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)  
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
```

- INSERT MEDIANTE SELECT

```
INSERT INTO first table name [(column1, column2, ... columnN)]  
  SELECT column1, column2, ...columnN  
  FROM second table name  
  [WHERE condition];
```

# UPDATE

- Debe usarse siempre where y UPDATE para seleccionar aquellos registros que se van actualizar, en caso contrario todos los registros serán actualizados

```
UPDATE table name  
SET column1 = value1, column2 = value2....., columnN = valueN  
WHERE [condition];
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
SQL> UPDATE CUSTOMERS  
SET ADDRESS = 'Pune'  
WHERE ID = 6;
```

# DELETE

- Debe usarse siempre where y DELETE para seleccionar aquellos registros que se van actualizar, en caso contrario todos los registros serán actualizados

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

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
SQL> DELETE FROM CUSTOMERS  
WHERE ID = 6;
```