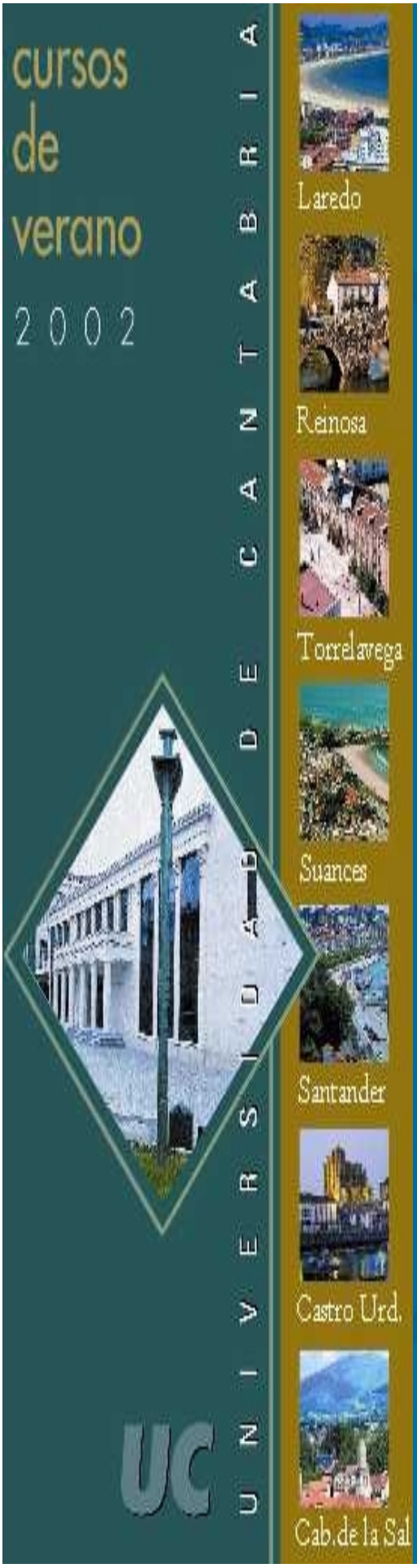


Curso SA.4.1

Java en el acceso Web a bases de datos

Introducción a los Sistemas de Gestión de Bases de Datos

Juan Manuel Cueva Lovelle
cueva@lsi.uniovi.es

Departamento de Informática
Universidad de Oviedo
OOTLab www.ootlab.uniovi.es

Contenidos

Tema 1º
Introducción

- ¿Qué es un SGBD?
- Partes de un SGBD
- Historia de los SGBD
- Problemas de los Sistemas de Procesamientos de Archivos
- Soluciones aportadas por los SGBD
- Informe ANSI/SPARC (1978)
- Clasificación de los SGBD
- ¿Qué es SQL?
- Historia de SQL
- SQL Estándar
- Sublenguajes de SQL

¿Qué es un SGBD ?

- *"Conjunto coordinado de programas, procedimientos, lenguajes,... Que suministra a los distintos tipos de usuarios los medios necesarios para describir y manipular los datos almacenados en la base, garantizando su seguridad"*

  [A. de Miguel et al. 1999]

Partes de un SGBD

- Un **SGBD** está formado por:
  - **Base de Datos (BD):** Colección de datos sobre un ente particular (empresa, organización, tema...).
  - **Programas *(Software*)** para acceder y manipular esos datos.
  - Otros elementos que pueden considerarse: ***Hardware*** **y Usuarios.**

Historia de los SGBD

- Los **SGBD** surgen, propiamente, a mediados de los años 60's, como un intento de mejorar el **Sistema de Procesamiento de Archivos** utilizado hasta entonces, de forma que una base de datos sea Integrada y Compartida:
  - **Integrada:** Unificación de archivos e información (evitando redundancias, mejorando los accesos...).
  - **Compartida:** Información utilizada por varios usuarios (total o parcialmente).

Problemas de los Sistema de Procesamiento de Archivos (I)

- **Redundancia** (datos duplicados) e **Inconsistencia** (datos contradictorios).
- **Dificultad de** Acceso **a ciertos datos o información:** Si no existen programas para acceder o calcular cierta información, no puede accederse a ella. Ej.: Calcular totales, o registros con cierta condición...
- **Aislamiento de Datos:** Los datos pueden estar en varios archivos con distintos formatos, que complican la creación de programas nuevos.
- **Falta de Integridad:** Es complicado mantener ciertas condiciones en la información. Ej.: Que el saldo sea superior a cierta cantidad, que un empleado esté adscrito a un número de Departamento que exista...

Problemas de los Sistema de Procesamiento de Archivos(II)

- **Problemas de Atomicidad en las operaciones:** A veces es esencial que para la consistencia de la BD se efectúen varias operaciones como si fueran una única operación, evitando que se produzcan fallos en medio de dicha operación. Ej.: En una transferencia bancaria hay de sacar el dinero de una cuenta y añadirlo a la otra.

- **Problemas en el Acceso Concurrente:** Si varios usuarios acceden a la vez a un dato pueden producirse errores. Ej.: Si se saca dinero de una misma cuenta desde dos sitios distintos.

- **Problemas de Seguridad:** Dificultad para controlar que ciertos usuarios no accedan a ciertos datos.

Soluciones aportadas por los SGDB (I)

Independencia de los Datos: Puede ser Física o Lógica.

- Es una de las grandes ventajas de los SGBD.
- **Objetivo: Que las aplicaciones no dependan de cómo se almacenen los datos físicamente o de su organización lógica.**

Puede modificarse la estructura de los datos sin que afecte a los programas. Ej.: Modificar nombres y formato de los ficheros, añadir o borrar campos...

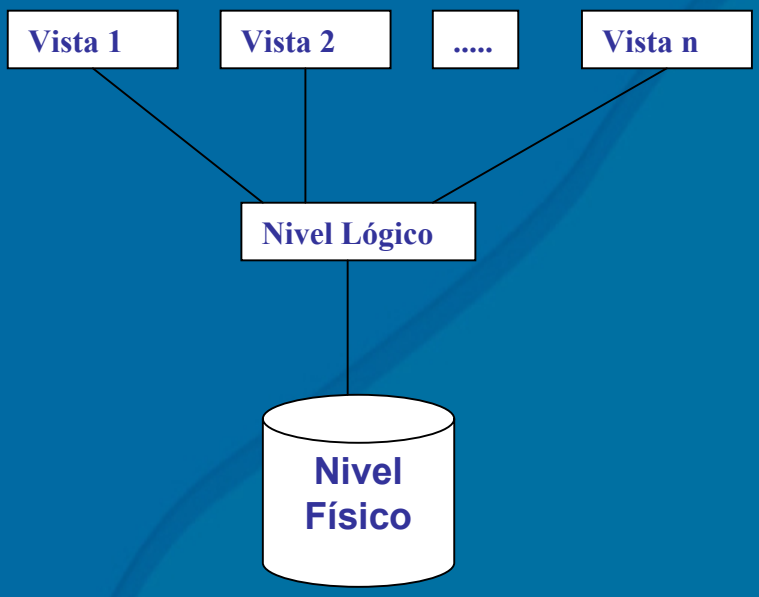
El **DBA** *(DataBase Administrator)* puede modificar esa estructura de los datos para mejorar el acceso y optimizar el uso de los recursos: Añadir o borrar índices, dividir en distintos archivos, usar nuevos discos u ordenadores (bases de datos distribuidas)...

Soluciones aportadas por los SGDB (II)

El concepto y las ventajas son similares al uso de TDA en programación.

- Esto implica que las aplicaciones (o los usuarios) tienen una **Visión Abstracta** de los datos: El SGBD esconde detalles sobre cómo se almacenan y mantienen los datos, de forma que facilita el acceso a estos datos, ya que sólo es necesario pedir al SGBD qué datos se requieren, sin especificar cómo conseguirlos.

Este es el principal objetivo de la **Arquitectura ANSI/SPARC (1978)** de 3 niveles:



Informe ANSI/SPARC (1978)
1. Nivel Externo o de vistas

- Mayor nivel de abstracción.
- Es el nivel más cercano a los usuarios y a su forma de ver los datos (su punto de vista y sus necesidades).
- Cada nivel externo describe sólo la parte de la BD que necesita para su aplicación o uso (de un programa, un usuario...).
- Esto facilita el control de los permisos de acceso (seguridad).
- Existen distintos tipos de usuarios: Programador en C (ve los datos como estructuras struct), usuario final de un programa...

Informe ANSI/SPARC (1978)
2. Nivel Conceptual o Lógico

- Conecta los otros dos niveles.
- Describe qué datos se almacenan en la BD y qué relaciones existen entre ellos.
- Los programadores de BD trabajan con este nivel, aunque también tienen su vista particular. Nunca un programa debe acceder directamente al nivel interno.
- **Independencia Lógica de los Datos:** Cuando modificar este nivel no implica la modificación de los niveles externos (programas).
- En muchas ocasiones la dependencia lógica de los programas es fuerte.

Informe ANSI/SPARC (1978)
3. Nivel Interno o Físico

- El más cercano al almacenamiento físico.
- Describe cómo se almacenan realmente los datos (estructuras de datos complejas de bajo nivel: Registros, ficheros, bytes, datos, organización...)
- Los DBA pueden conocer y controlar ciertos aspectos de este nivel, pero no es necesario: Puede conseguirse optimizar el sistema y ganar en eficiencia *(tuning)*.
- **Independencia Física de los Datos:** Cuando modificar este nivel no implica la modificación de los niveles externos (programas).

Clasificación de los SGBD

- Dependiendo del **Modelo de Datos** utilizado:
  - **Modelo Relacional**: Un conjunto de tablas.
  - **Modelo Orientado a Objetos**: Un conjunto de objetos.
  - Otros modelos: **Jerárquico** y en **Red** (CODASYL).
- Dependiendo del **Número de Usuarios** que soporta:
  - **Monousuario**
  - **Multiusuario.**
- Dependiendo del **Número de Sitios** en los que está la BD:
  - **SGBD Centralizado.**
  - **SGBD Distribuido** (y conectado mediante una red): Pueden ser **homogéneos** (con mismo software en todos los lugares) y **heterogéneos**.
- Estándares de comunicaciones con SGBD:
  - **ODBC** *(Open Database Connectivity)*
  - **ADO** (ActiveX Data Objects)
  - **JDBC** (Java DataBase Connectivity)

¿Qué es SQL ?

- **SQL** (Structured Query Language), **lenguaje de consulta estructurado**.
- Es un lenguaje de programación para el manejo de sistemas de gestión de bases de datos relacionales.
- La versión original fue desarrollada en la década de los 70 por IBM para sistemas de gestión de bases de datos relacionales.
- En 1986 el American National Standards Institute (ANSI) publicó la primera definición definición de SQL estándar.

Historia de SQL (I)

- Junio de 1970
  - E. F. Codd publica el artículo "A Relational Model of Data for Large Shared Data Banks", en la revista *Communications of the ACM (*Association of Computer Machinery, www.acm.org) .
  - Actualmente el modelo de Codd's es la base de los sistemas de gestión de bases de datos relacionales (SGBDR o las siglas en lengua inglesa RDBMS).
  - Los laboratorios de IBM en San José California implementan los prototipos de los estudios de Codd’s
- 1974-75
  - Nace SEQUEL (Structured English QUEry Language) implementado por IBM.
- 1976-77
  - Evolución a SEQUEL/2 que posteriormente IBM denomina SQL

Historia de SQL (II)

- 1979
  - Relational Software, Inc. (ahora Oracle Corporation) introduce el primer SGBDR basado en SQL
- Posteriormente surgen otros productos SQL
  - DB2, SYBASE, INFORMIX, INTERBASE,...
- El lenguaje se convierte en un estándar de facto, aunque con múltiples variantes según los distintos fabricantes

SQL Estándar (I)

- 1982
  - El Comité de Bases de Datos X3H2 de ANSI (American National Standards Institute) presenta un lenguaje relacional estándar basado principalmente en el SQL propio de los productos de IBM
- 1986
  - Se aprueba la norma SQL/ANSI
- 1987
  - Se aprueba la norma SQL/ISO
- 1989
  - Se revisa el estándar añadiéndose algunos aspectos de integridad referencial
  - Esta revisión se denomina ***Addendum***
- 1992
  - Se aprueba una nueva revisión conocida como SQL-92. Los nombres formales son:
    - ANSI X3.135-1992, "Database Language SQL"
    - ISO/IEC 9075:1992, "Database Language SQL"

SQL Estándar (II)

- SQL-92 define cuatro niveles de conformidad (de menor a mayor exigencia):
  - *Entry*
  - *Transitional*
  - *Intermediate*
  - *Full*
- Una implementación conforme el estándar SQL-92 debe soportar al menos el nivel *Entry*

Sublenguajes de SQL

- **DDL** (Data Description Language), **lenguaje de definición** de datos, incluye órdenes para definir, modificar o borrar las tablas en las que se almacenan los datos y de las relaciones entre estas. Es el que más varia de un sistema a otro.
- **DCL** (Data Control Language), **lenguaje de control** de datos, contiene elementos útiles para trabajar en un entorno multiusuario, en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.
- **DML** (Data Manipulation Language), **lenguaje de manipulación** de datos, nos permite recuperar los datos almacenados en la base de datos y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo nuevos datos, suprimiendo datos antiguos o modificando datos previamente almacenados.

Tema 2º: DDL
Lenguaje de definición de datos

- Introducción
- Bases de datos
- Tablas
- Filas o “tuplas”
- Columnas
- Índices
- Sinónimos
- Vistas

DDL

Lenguaje de definición de datos

- Sublenguaje utilizado para describir los atributos de una base de datos, especialmente las tablas, campos, índices y la estrategia de almacenamiento
- Este sublenguaje es el encargado de la definición y mantenimiento de aquellos elementos SQL que contienen la información.
- Dichos elementos son los siguientes:
  - **Bases de datos**
  - **Tablas**
  - **Filas o "tuplas"**
  - **Columnas**
  - **Índices**
  - **Sinónimos**
  - **Vistas**

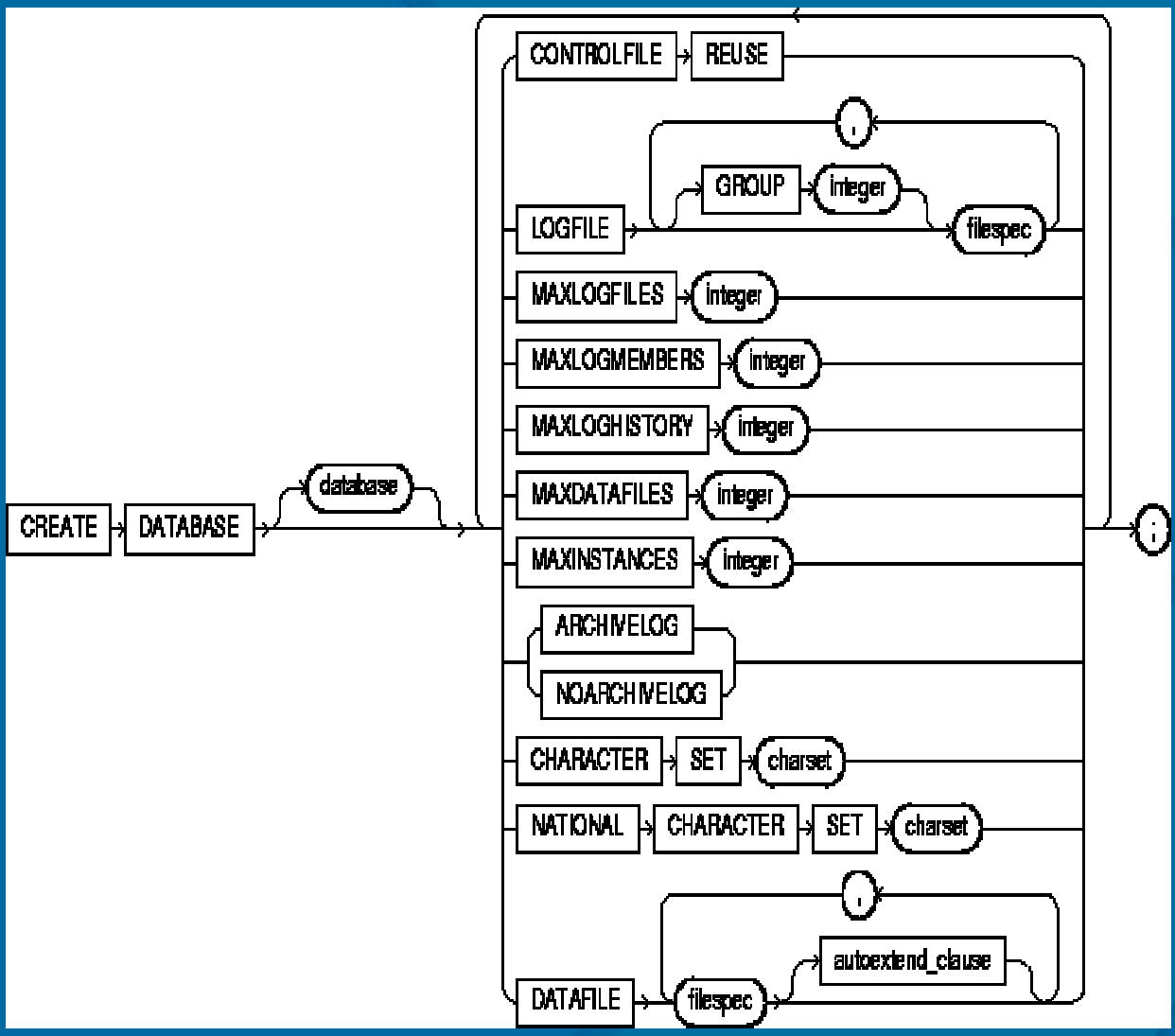
Instrucciones DDL

- Crear, modificar y eliminar elementos y estructuras de la base de datos, incluyendo la propia base da datos y los usuarios (CREATE, ALTER, DROP)
- Cambiar los nombres de los distintos elementos (RENAME)
- Borrar todos los datos sin modificar la estructura (TRUNCATE)
- Reunir estadísticas acerca de elementos de los esquemas, validar la estructura de los distintos elementos (ANALYZE)
- Conceder y quitar privilegios (GRANT, REVOKE)
- Opciones de auditoria (AUDIT, NOAUDIT)
- Añadir comentarios al diccionario de datos (COMMENT)

Bases de Datos

- Las operaciones que se pueden realizar sobre una base de datos son:
  - Creación de la Base de Datos
  - Borrado de la Base de Datos

Creación de la Base de Datos
CREATE DATABASE



Borrado de la Base de Datos

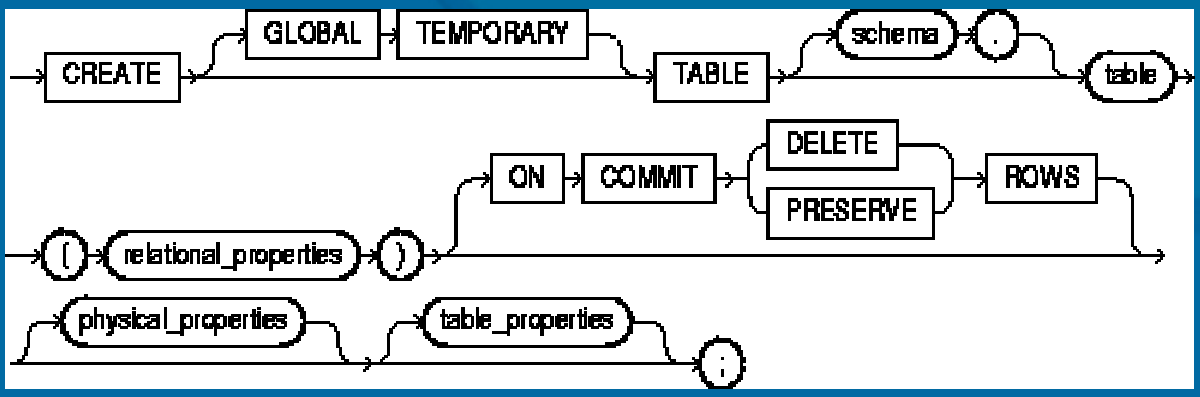
DROP DATABASE LINK

OPERACIONES CON TABLAS

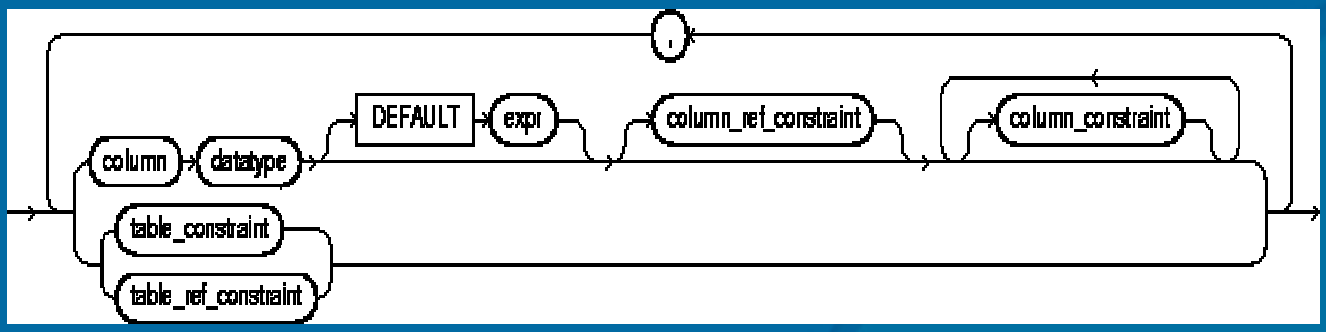
- Las operaciones que se pueden realizar sobre las tablas son:
  - Crear tablas (CREATE TABLE)
  - Modificar tablas (ALTER TABLE)
  - Borrar tablas (DROP TABLE)

CREAR TABLAS

CREATE TABLE



relational_properties::=

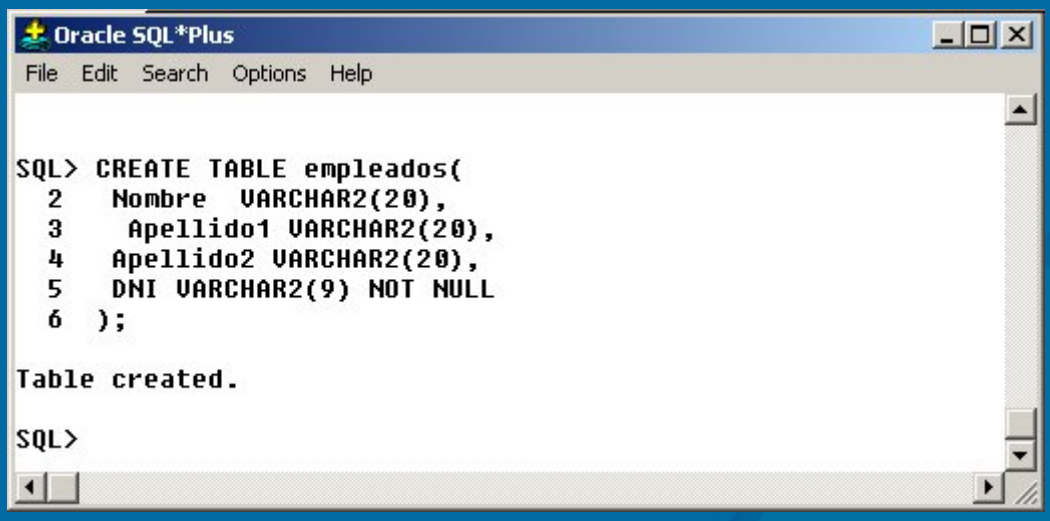


Forma de uso habitual:

CREATE TABLE <nombre_tabla> (
<nombre_columna1> <tipo_dato> <[not] null>,
<nombre_columna2> <tipo_dato> <[not] null>,
...
) ;

CREAR TABLAS
Ejemplo (I)

```
CREATE TABLE empleados(
        Nombre   VARCHAR2(20),
        Apellido1 VARCHAR2(20),
        Apellido2 VARCHAR2(20),
        DNI VARCHAR2(9) NOT NULL
);
```



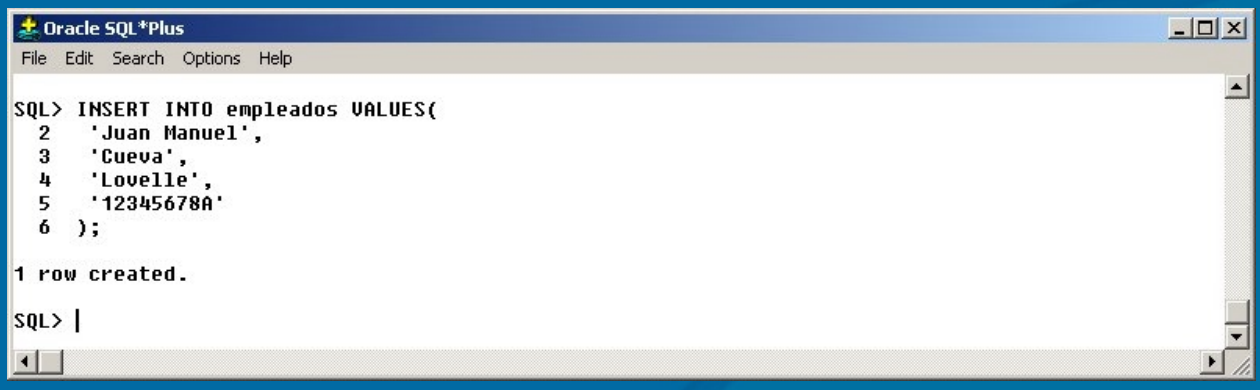
•**VARCHAR2** – Es un tipo de datos de Oracle 8 para almacenar cadenas de caracteres. Entre paréntesis se coloca el número máximo de caracteres que puede contener esta columna. Así VARCHAR2(20) puede contener hasta 20 caracteres. En Oracle 8 el valor máximo del tamaño de una cadena definida con VARCHAR2 es 4000.

INSERTAR DATOS EN TABLAS

(esta instrucción pertenece a DML)

Ejemplo

```
INSERT INTO empleados VALUES(
        'Juan Manuel',
        'Cueva',
        'Lovelle',
        '12345678A'
);
```



MOSTRAR TODAS LAS COLUMNAS DE UNA TABLA

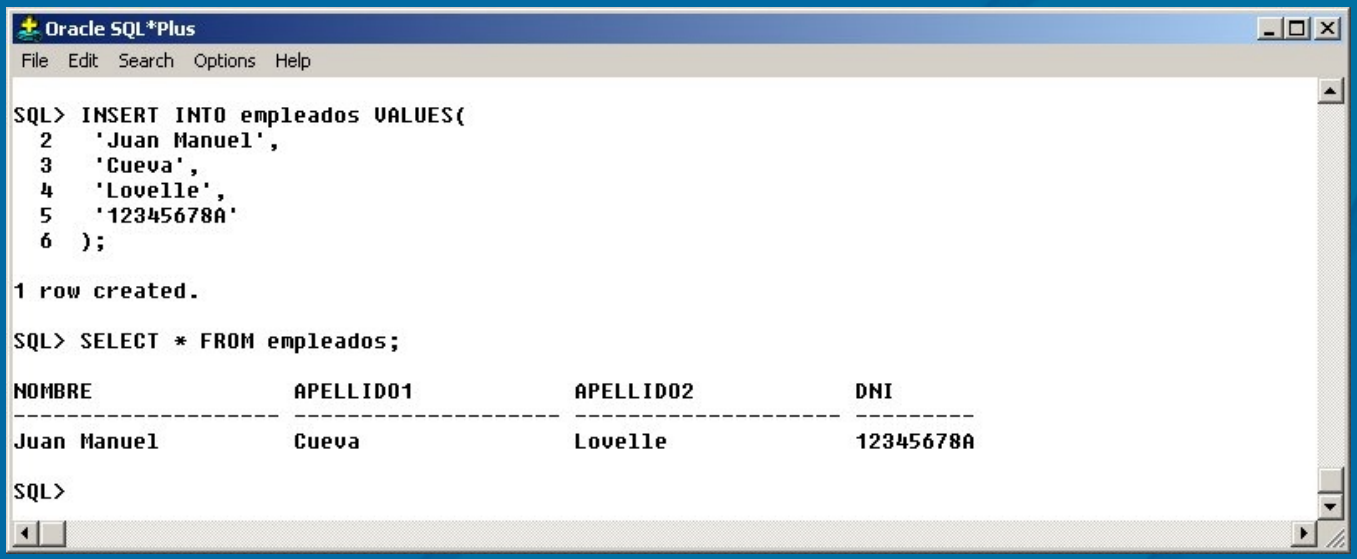
(esta instrucción pertenece a DML)

Ejemplo (I)

```
SELECT * FROM empleados;
```

Resultado:

```
NOMBRE               APELLIDO1            APELLIDO2            DNI
-------------------- -------------------- -------------------- ---------
Juan Manuel          Cueva                Lovelle              12345678A
```



MOSTRAR ALGUNAS COLUMNAS DE UNA TABLA

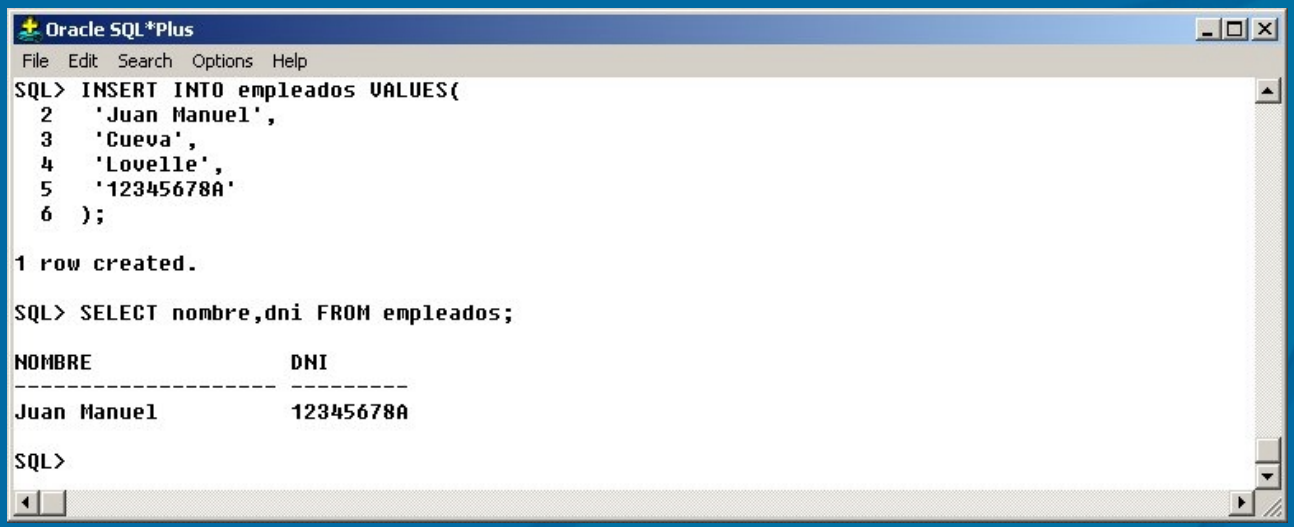
(esta instrucción pertenece a DML)

Ejemplo (II)

```
SELECT nombre,dni FROM empleados;
```

Resultado:

```
NOMBRE               DNI
-------------------- ---------
Juan Manuel          12345678A
```



CREAR TABLAS

Ejemplo (II) UNIQUE

```
CREATE TABLE empleados(
        Nombre   VARCHAR2(20),
        Apellido1 VARCHAR2(20),
        Apellido2 VARCHAR2(20),
        DNI VARCHAR2(9) NOT NULL UNIQUE
);
```

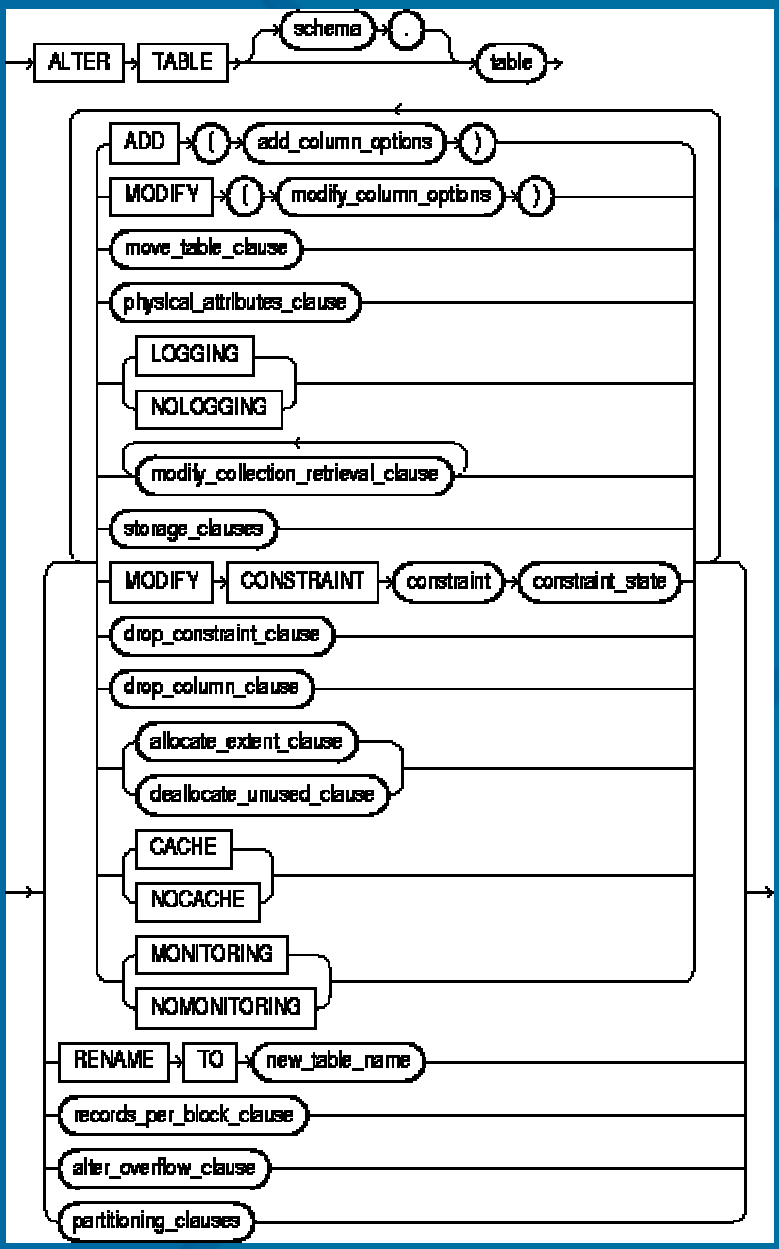
Oracle SQL*Plus

File Edit Search Options Help

```
SQL> CREATE TABLE empleados(
  2   Nombre  VARCHAR2(20),
  3    Apellido1 VARCHAR2(20),
  4   Apellido2 VARCHAR2(20),
  5   DNI VARCHAR2(9) NOT NULL UNIQUE
  6  );

Table created.

SQL> INSERT INTO empleados VALUES(
  2   'Juan Manuel',
  3   'Cueva',
  4   'Lovelle',
  5   '12345678A'
  6  );

1 row created.

SQL> INSERT INTO empleados VALUES(
  2   'Juan Manuel',
  3   'Cueva',
  4   'Lovelle',
  5   '12345678A'
  6  );
INSERT INTO empleados VALUES(
*
ERROR at line 1:
ORA-00001: unique constraint (SYS.SYS_C001222) violated


SQL>
```

MODIFICAR TABLAS

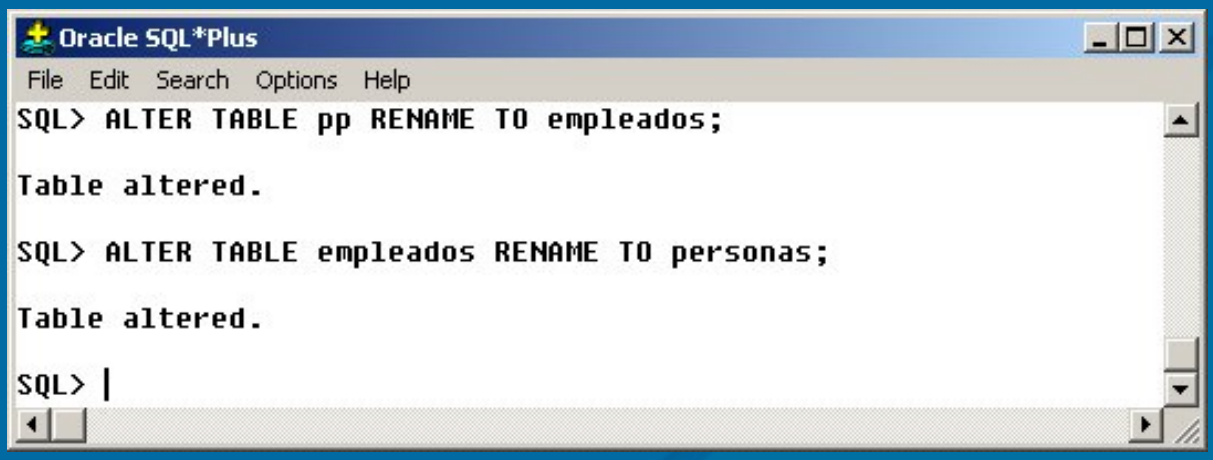
Cambia la definición de una tabla existente ALTER TABLE



MODIFICAR TABLAS
Ejemplo (I)

- **Cambiar el nombre de la tabla**

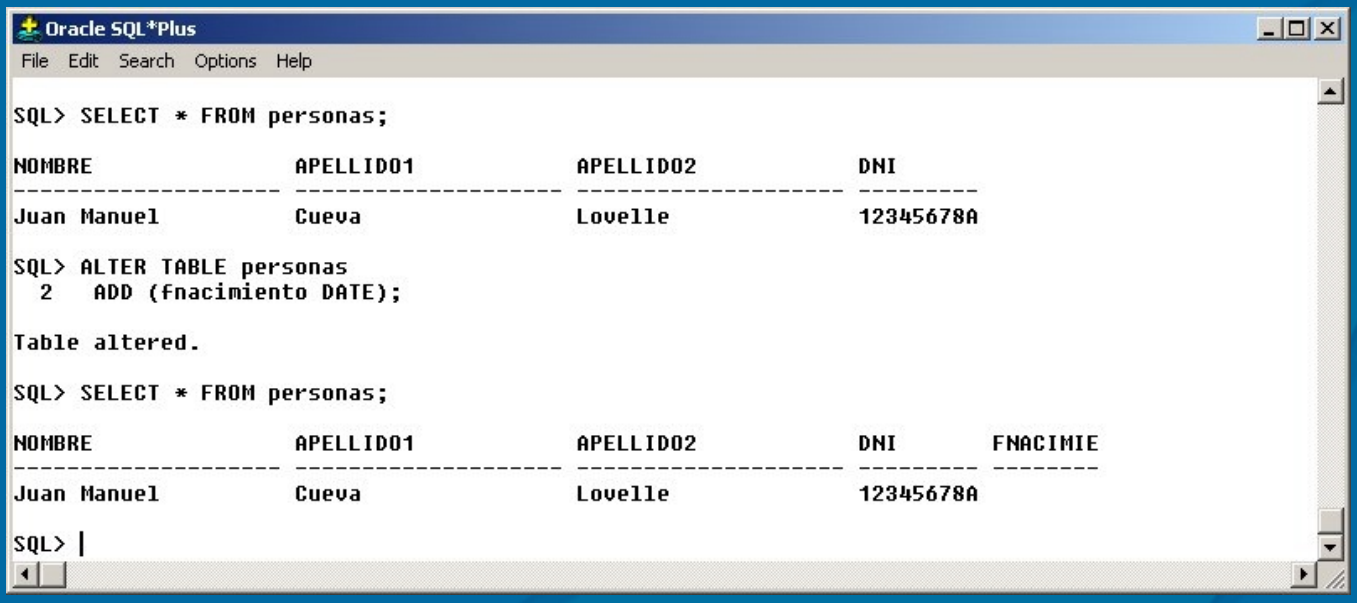
```
ALTER TABLE empleados
      RENAME TO personas;
```



MODIFICAR TABLAS
Ejemplo (II)

- **Añadir una columna a una tabla**

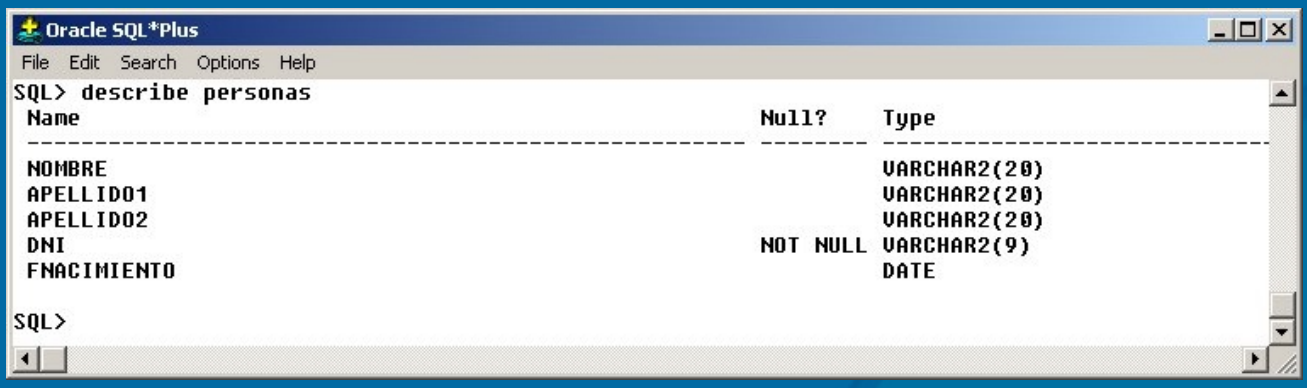
```
ALTER TABLE personas
      ADD (fnacimiento DATE);
```



VISUALIZAR LA ESTRUCTURA DE UNA TABLA (Sólo en Oracle SQL*PLUS)

- **Visualiza la estructura de una tabla**

```
DESCRIBE personas;
```

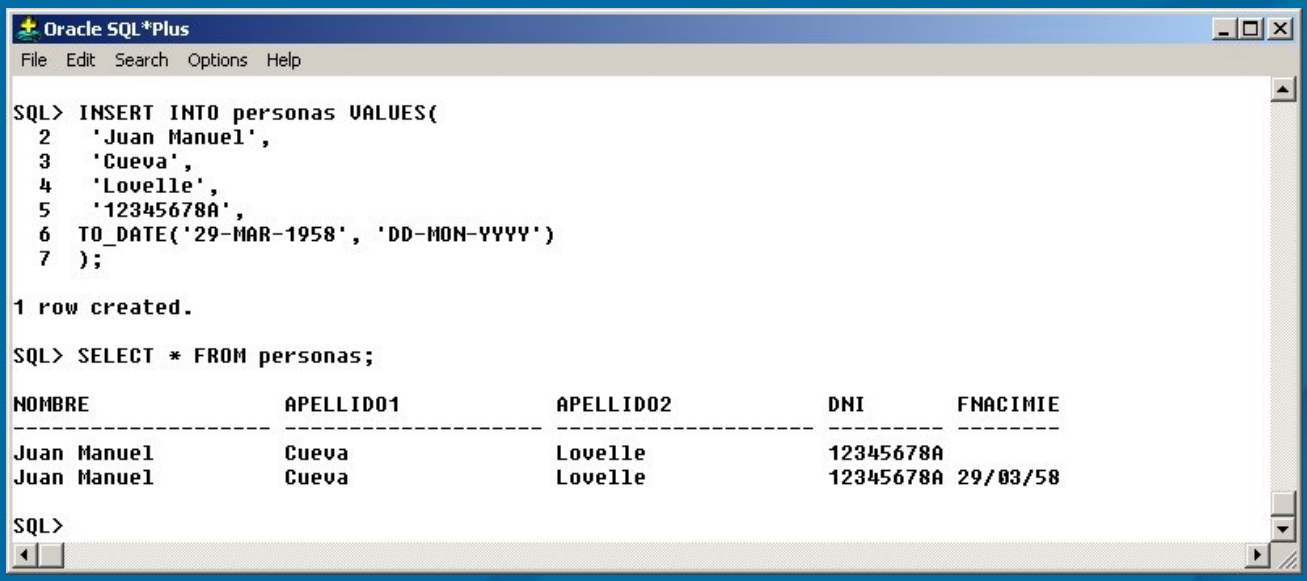


INSERTAR DATOS EN TABLAS

(esta instrucción pertenece a DML)

Ejemplo

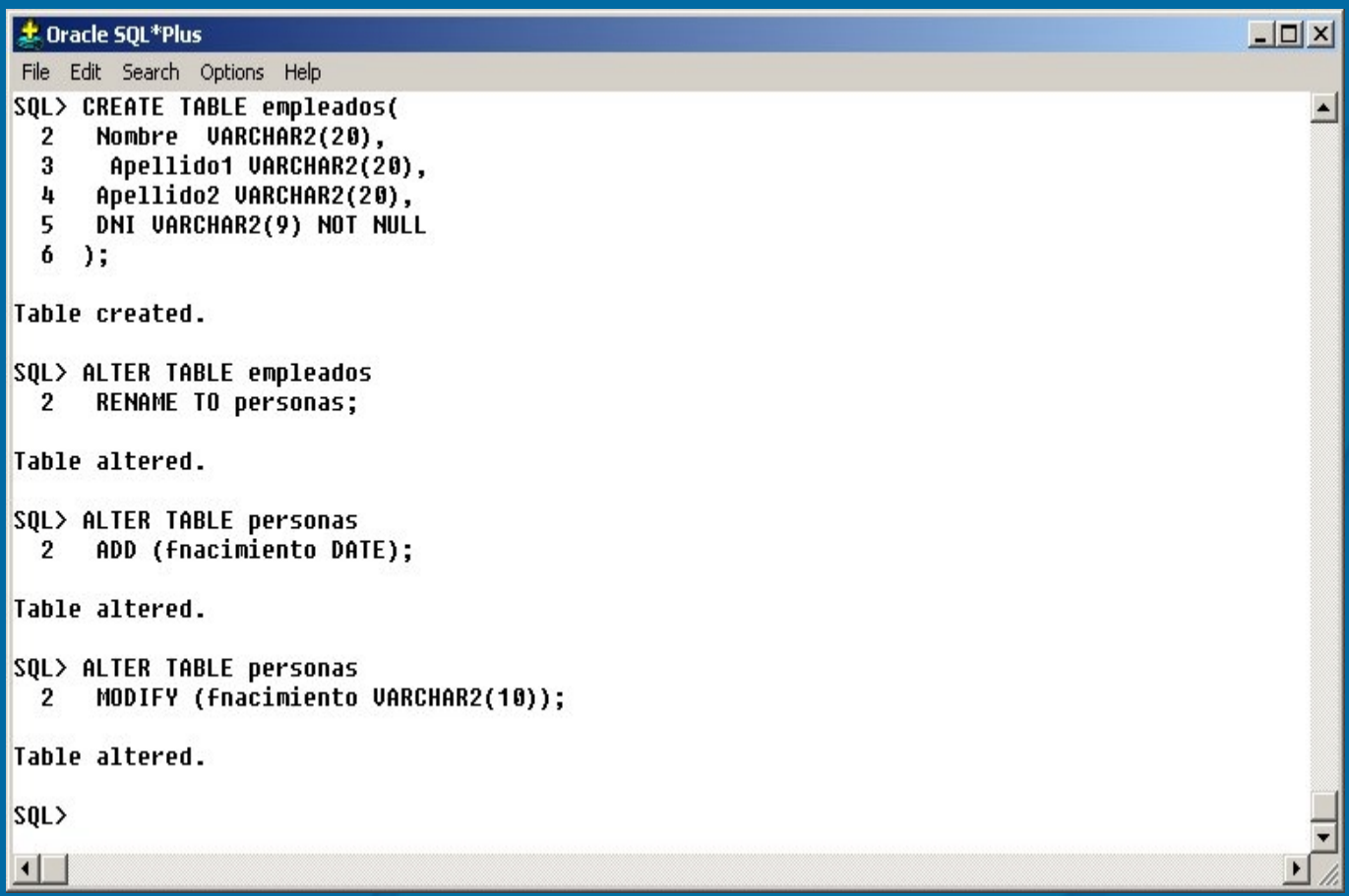
```
INSERT INTO personas VALUES(
        'Juan Manuel',
        'Cueva',
        'Lovelle',
        '12345678A',
TO_DATE('29-MAR-1958', 'DD-MON-YYYY')
);
```



MODIFICAR TABLAS
Ejemplo (III)

- **Modificar el tipo de una columna en una tabla**
  - **Debe estar vacía la columna a la que se desea modificar el tipo**

```
ALTER TABLE personas
 MODIFY (fnacimiento VARCHAR2(10));
```



MODIFICAR TABLAS
Ejemplo (IV)

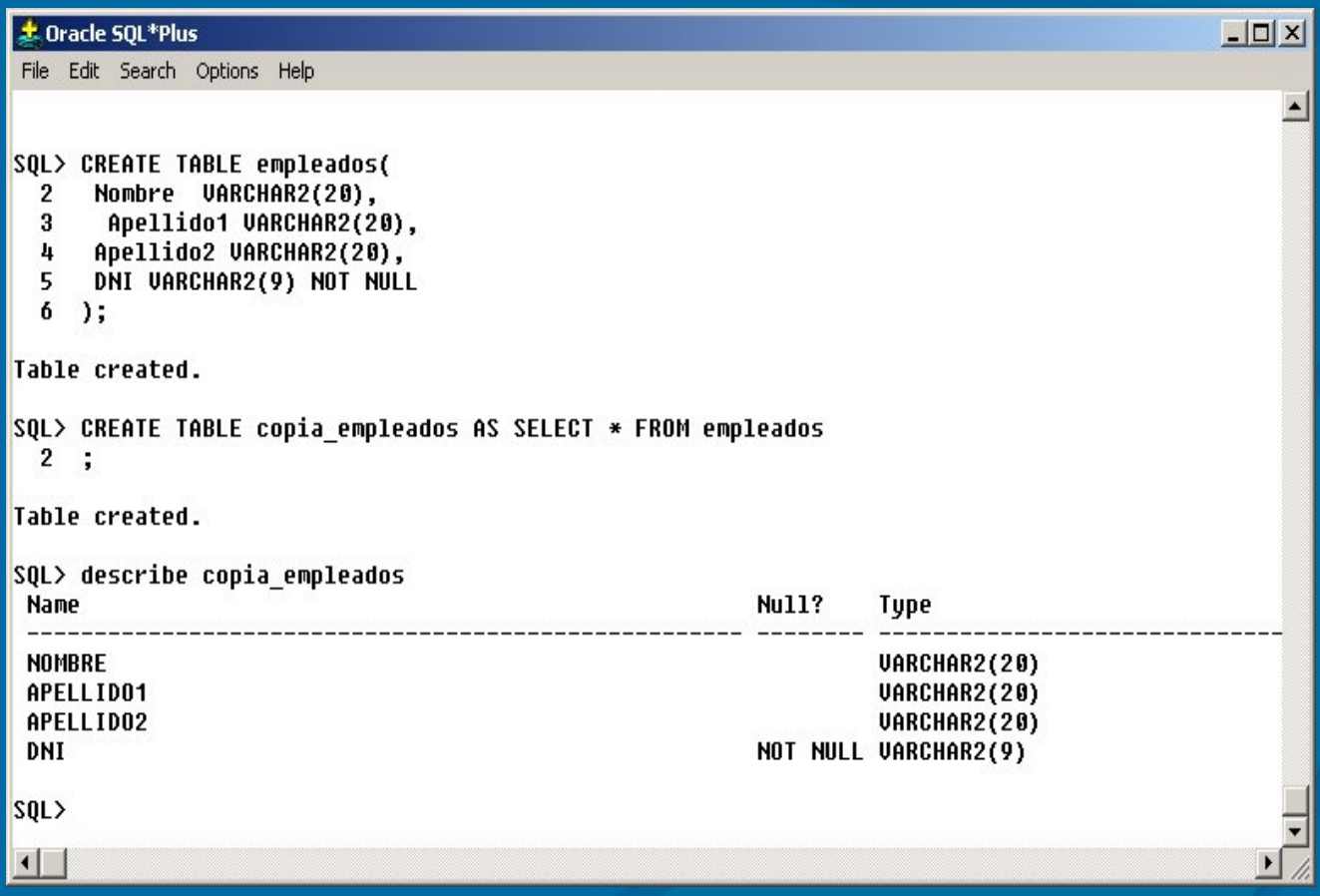
- **Eliminar una columna en una tabla**

```
ALTER TABLE personas
      DROP (fnacimiento);
```

CREAR TABLAS COMO RESULTADO DE UNA CONSULTA
Ejemplo (V)

- **Copiar una tabla en otra**

```
CREATE TABLE copia_empleados AS SELECT * FROM empleados;
```



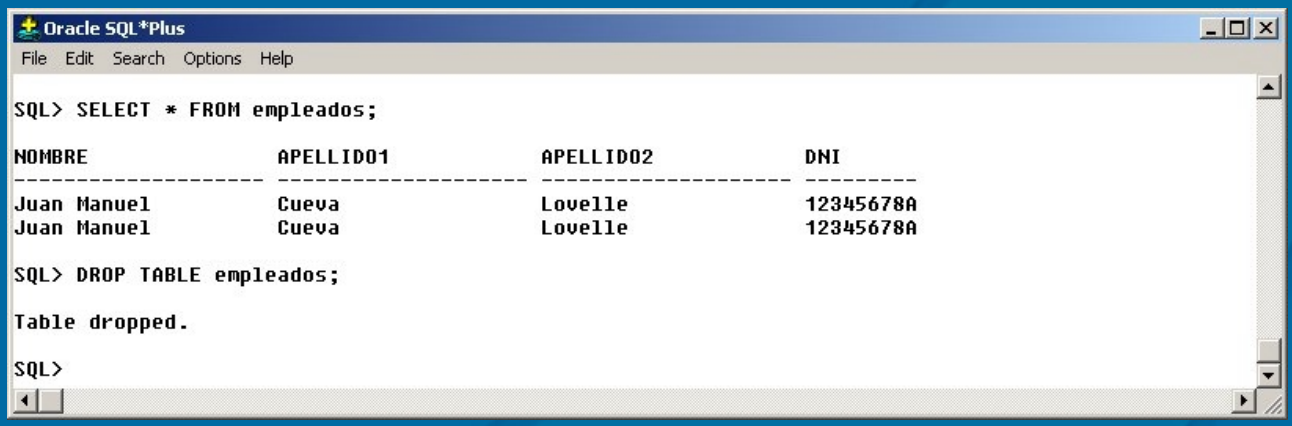
BORRAR TABLAS

Borra una tabla existente

DROP TABLE

Forma de uso habitual:

DROP TABLE <nombre_tabla> ;



TIPO DE DATOS

NUMBER

- NUMBER (t,d) Tipo numérico para enteros y reales.
  - t = número total de dígitos. Rango 1-38.
  - d = número de decimales. Rango –84 a +127
  - Ejemplo: NUMBER(5,2) sólo puede admitir números menores de 999.99
  - Tipos de datos derivados de NUMBER son INTEGER, DECIMAL, SMALLINT y REAL

Ejemplos

7456123.89	NUMBER	7456123.89
7456123.89	NUMBER(9)	7456124
7456123.89	NUMBER(9,2)	7456123.89
7456123.89	NUMBER(9,1)	7456123.9
7456123.89	NUMBER(6)	Excede la precisión
7456123.89	NUMBER(7,-2)	7456100
7456123.89	NUMBER(-7,2)	Error

INSERTANDO DATOS NUMBER

```
INSERT INTO empleados VALUES(
        'Juan Manuel',
        'Cueva',
        'Lovelle',
        '12345678A',
TO_DATE('29-MAR-1958', 'DD-MON-YYYY'),
        250000,
        1502.532
);
INSERT INTO empleados VALUES(
        'Antonio',
        'Cueva',
        'Fernández',
        '02345678A',
TO_DATE('22-FEB-1991', 'DD-MON-YYYY'),
        550000,
        11502.5334
);
INSERT INTO empleados VALUES(
        'Paloma',
        'Cueva',
        'Fernández',
        '01345678A',
        '30/03/93',
        NULL,
        11502.53347
);
```

EJERCICIO

Crear una tabla con tipos de datos numéricos

```
CREATE TABLE empleados(
       Nombre  VARCHAR2(20),
       Apellido1 VARCHAR2(20),
       Apellido2 VARCHAR2(20),
       dni VARCHAR2(9) NOT NULL,
       fnacimiento DATE,
       salarioPtas NUMBER(10),
       salarioEuros NUMBER (8,3)
);
```

INTEGRIDAD REFERENCIAL

- El objetivo de la integridad referencial es evitar la inconsistencia de datos entre las tablas de la base de datos
- La integridad referencial se realiza con la definición de las claves primarias y claves referenciales que controlarán la integridad entre todas las tablas de la base de datos

INTEGRIDAD REFERENCIAL
Definición de la clave primaria **(PRIMARY KEY)**

- La definición y borrado de la clave primaria se encuentra implícita respectivamente en la sintaxis de las instrucciones CREATE TABLE y ALTER TABLE
- La columna o columnas que compongan la clave primaria deberán tener asignado el atributo NOT NULL
- La definición de la clave primaria implica la definición de un índice único que estará compuesto por dichas columnas

INTEGRIDAD REFERENCIAL

Definición de la clave primaria (PRIMARY KEY)

Ejemplo usando CREATE TABLE

```
CREATE TABLE personal(
       Nombre  VARCHAR2(20),
       Apellido1 VARCHAR2(20),
       Apellido2 VARCHAR2(20),
       dni VARCHAR2(9) NOT NULL,
       fnacimiento DATE,
       salarioPtas NUMBER(10),
       salarioEuros NUMBER (8,3),
CONSTRAINT cp_dni PRIMARY KEY(dni));
```

INTEGRIDAD REFERENCIAL

Definición de la clave primaria
(PRIMARY KEY)

Ejemplo usando ALTER TABLE

```
CREATE TABLE empleados(
       Nombre  VARCHAR2(20),
       Apellido1 VARCHAR2(20),
       Apellido2 VARCHAR2(20),
       dni VARCHAR2(9) NOT NULL,
       fnacimiento DATE,
       salarioPtas NUMBER(10),
       salarioEuros NUMBER (8,3)
);
```

- **Añadir una clave primaria**

```
ALTER TABLE empleados
      ADD PRIMARY KEY (dni);
```

- **Eliminar una clave primaria**

```
ALTER TABLE empleados
      DROP PRIMARY KEY;
```

INTEGRIDAD REFERENCIAL
Definición de las claves referenciales
(FOREIGN KEY)

- La definición de las claves referenciales se encuentra implícita respectivamente en la sintaxis de las instrucciones CREATE TABLE y ALTER TABLE
- La instrucción ALTER TABLE permite asimismo el borrado de claves referenciales
- La definición de una clave referencial no supone la creación de un índice
- Una clave referencial tiene que estar compuesta por el mismo número de columnas que la respectiva clave primaria en la tabla referenciada.
- Las columnas tienen que especificarse en el mismo orden y ser del mismo tipo de dato SQL

INTEGRIDAD REFERENCIAL
Definición de las claves referenciales (FOREIGN KEY)
Ejemplo de CREATE TABLE

- **TABLA PADRE O TABLA REFERENCIADA**

```
CREATE TABLE departamentos(
        CodDep NUMBER(3) PRIMARY KEY,
        Nombre  VARCHAR2(20),
        Localizacion VARCHAR2(20)
);
```

- **TABLA HIJA O TABLA DEPENDIENTE**

```
CREATE TABLE trabajadores (
        dni NUMBER(8) PRIMARY KEY,
        nombre VARCHAR2(20),
        apellido1 VARCHAR2(20),
        apellido2 VARCHAR2(20),
        CodDep NUMBER(3)NOT NULL
CONSTRAINT cr_departamentos REFERENCES
departamentos);
```

INTEGRIDAD REFERENCIAL
Definición de las claves referenciales (FOREIGN KEY)
Ejemplo de ALTER TABLE

- **TABLA PADRE O TABLA REFERENCIADA**

```
CREATE TABLE depto(
        CodDep NUMBER(3),
        Nombre  VARCHAR2(20),
        Localizacion VARCHAR2(20)
);

ALTER TABLE depto
        ADD PRIMARY KEY (CodDep);
```

- **TABLA HIJA O TABLA DEPENDIENTE**

```
CREATE TABLE empleo (
        dni NUMBER(8) PRIMARY KEY,
        nombre VARCHAR2(20),
        apellido1 VARCHAR2(20),
        apellido2 VARCHAR2(20),
        CodDep NUMBER(3)NOT NULL
);

ALTER TABLE empleo
      ADD FOREIGN KEY (CodDep) REFERENCES
depto(CodDep);
```

INTEGRIDAD REFERENCIAL
Definición de las claves referenciales (FOREIGN KEY)
Ejemplo

- **Insertando datos en la TABLA PADRE**

```
INSERT INTO depto VALUES(10,'Informática',
'Oviedo');
```

1 row created.

- **Insertando datos en la TABLA HIJA**

```
INSERT INTO empleo VALUES (12345678, 'Guillermo',
'Cueva', 'Fernández',10);
```

1 row created.

```
INSERT INTO empleo VALUES (2345678, 'Antonio',
'Cueva', 'Fernández',11);
```

ORA-02291: integrity constraint (SYS.SYS_C001243) violated - parent key not found

EJERCICIO

- La Comunidad Autónoma del Principado de Asturias desea construir una base de datos con las casas rurales.
- Una casa rural se identifica por un nombre (“Casa Pepín”,”Les buenes fabes”,....) y una dirección, teléfono, fax, e-mail y página web. Así como los datos de la persona de contacto.
- En cada casa rural trabajan una serie de personas que se identifican por su NIF y sus datos personales.
- Las casas rurales se alquilan por número de personas que los pueden ocupar. Cada casa rural tiene un precio, que cambia según la época del año y el número de personas.
- Las casas rurales proponen actividades multi-aventura a los huéspedes (senderismo, piragüismo,...) siguiendo una ruta determinada que se clasifica por el grado de dificultad de 1 a 10. Estas actividades se realizan unos días, horas y épocas determinadas del año.

TEORÍA DE LA NORMALIZACIÓN

- La primera aproximación en el diseño de una base de datos, produce un esquema relacional estructurado y con redundancias.
- Siempre se deben aplicar un conjunto de reglas denominadas “teoría de la normalización” que nos permiten asegurar que un esquema relacional cumple unas propiedades.

TEORÍA DE LA NORMALIZACIÓN
Problemas

- Incapacidad para almacenar ciertos hechos
- Redundancia y posibilidad de inconsistencias
- Ambigüedades
- Pérdida de información
- Pérdida de ciertas restricciones de integridad que dan lugar a interdependencias entre los datos (dependencias funcionales)
- Por la redundancias de datos aparecen estados no válidos en el mundo real. Se denominan: anomalías de inserción, borrado y modificación

TEORÍA DE LA NORMALIZACIÓN
Problemas Ejemplo (1)

- Relación ***imparte:*** almacena datos sobre los **profesores**(Profesor, Categoría) y sobre las **asignaturas que imparten** (Asignatura, Titulación y curso)

- **Ejemplo de diseño inadecuado**

Nom_Profesor	Categoría	Nom_Asignatura	Titulación	Curso
Cueva, JM	Catedrático EU	Metodología de la Programación II	Ingeniero Técnico en Informática Oviedo	2
Cueva, JM	Catedrático EU	Procesadores de Lenguaje	Ingeniero en Informática	4
Juan Fuente,A	Titular	Procesadores de Lenguaje	Ingeniero en Informática	4
Pepón, José	Catedrático EU	Metodología de la Programación II	Ingeniero Técnico en Informática Gijón	2

TEORÍA DE LA NORMALIZACIÓN
Problemas Ejemplo (2)

- Los problemas de esta relación se derivan de la ***redundancia***. Por ejemplo: la categoría del profesor se repite por cada asignatura que imparte.
- Esta redundancia produce:
  - ***Anomalías de inserción***: para dar de alta una asignatura se deben insertar tantas tuplas como profesores la impartan.
  - ***Anomalías de modificación:*** ya que para cambiar el nombre de la titulación, se deben modificar todas las tuplas que corresponden a la titulación.
  - ***Anomalías de borrado***: ya que el borrado de una asignatura obliga a borrar todas las tuplas que se refieren a esa titulación.

PRINCIPIO BÁSICO DE TODO DISEÑO

- *"Hechos distintos se deben almacenar en objetos distintos"*
- Para evitar las dudas respecto a si un determinado esquema relacional es o no correcto, es preferible aplicar un método formal de análisis, que aplicado a dicho esquema permita obtener otro esquema que cumpla ciertos requisitos.
- Este método formal de análisis es:

La teoría de la normalización

Solución del Ejemplo

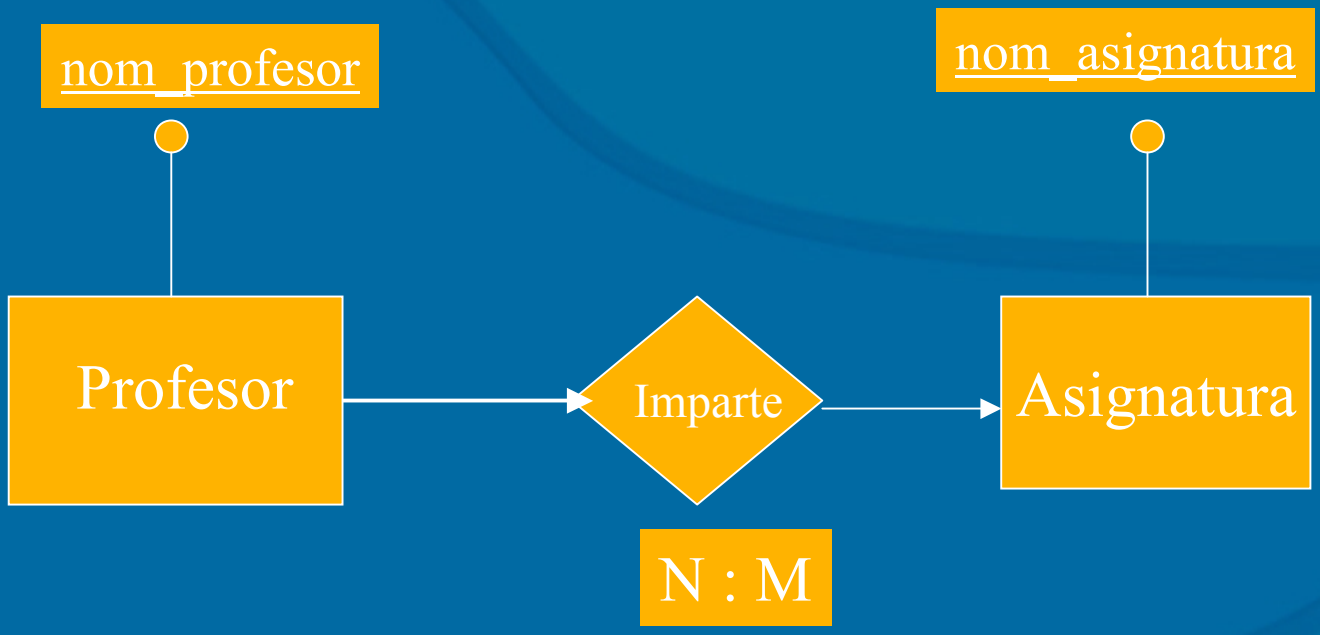
- La teoría de la normalización evita las redundancias y las anomalías de actualización.
- Se obtienen relaciones más estructuradas.
- Solución de la relación ***Imparte***:

  ASIGNATURA(nom_asignatura, titulación, curso)

  PROFESOR(nom_profesor, categoría)

  IMPARTE(nom_asignatura, nom_profesor)
- Se han separado los hechos distintos en relaciones distintas.

Modelo Entidad /Relación



Modelo Relacional

PROFESOR(nom_profesor, categoría)

IMPARTE(nom_asignatura, nom_profesor)

ASIGNATURA(nom_asignatura, titulación, curso)

El proceso de normalización

- La teoría de la normalización se centra en las ***formas normales***
- Un esquema de relación está en una determinada forma normal si satisface un conjunto específico de restricciones

Primera Forma Normal (1FN)

- Una relación está en 1FN si los valores de la relación son atómicos para cada atributo de la relación
- Prohibición de que en una relación existan grupos repetitivos.
- Un atributo no puede tomar mas de un valor del dominio subyacente.
- Su cumplimiento es **obligatorio** en el modelo relacional.
- Cualquier tabla de dos dimensiones y con elementos atómicos se dice que está en primera forma normal.

Relación que no está en 1FN

Finca	Propietario	Situación	Límites
1001	J.M. Cueva	Oviedo	**123, 234, 456, 567**
1002	J.M. Lovelle	Oviedo	**223, 234, 123, 45**
1001	J.M. García	Oviedo	**345, 445, 555, 555**
. . .	. . .	. . .	. . .

Segunda Forma Normal (2FN)

- Una tabla esta en segunda forma normal cuando
  - Está en primera forma normal (1FN)
  - Todos los atributos que no son claves dependen por completo del atributo que es clave
- Otra definición:
  - Está en primera forma normal (1FN)
  - Cada atributo no principal tiene dependencia funcional completa respecto de cada una de las claves

Relación que no está en 2FN

ASIGNACIÓN

ID_TRABAJADOR	ID_EDIFICIO	FECHA_INICIO	NOMBRE
1010	912	01/02/99	JM Cueva
1011	934	03/02/98	JM Cueva
1015	999	02/02/97	JM García
1017	100	08/12/99	G Lovelle

- NOMBRE está determinado por ID_TRABAJADOR
- Dejar esta relación en esta forma puede conducir a los siguientes problemas
  - El nombre del trabajador se repite en cada fila que se refiera a una asignación para ese trabajador
  - Si el nombre del trabajador cambia, cada fila que se refiera a una asignación de ese trabajador debe actualizarse. Esto es una anomalía de actualización
  - Debido a esta redundancia los datos podrían convertirse en inconsistentes, mostrando diferentes nombres para el mismo ID_TRABAJADOR

SOLUCIÓN

- **Se resuelve con dos PROYECCIONES**

  - **ASIGNACIÓN** (ID_TRABAJADOR, ID_EDIFICIO, FECHA_INICIO)

    Clave externa ID_TRABAJADOR referencia a TRABAJADOR

  - **TRABAJADOR** (ID_TRABAJADOR, NOMBRE)

- La 2FN reduce la redundancia y la inconsistencia

Tercera Forma Normal (3FN)

- Una tabla esta en tercera forma normal cuando
  - Está en segunda forma normal (2FN)
  - No tiene dependencias transitivas
- Dependencia transitiva: aparece cuando un atributo no clave es funcionalmente dependiente de uno o mas atributos no claves.
- Otra definición:
  - Está en segunda forma normal (2FN)
  - Los atributos que no forman parte de ninguna clave facilitan información sólo acerca de la/s clave/s y no acerca de otros atributos.

Relación que no está en 3FN

- **SOCIO** (cod_socio, ciudad, país)
- Tiene las dependencias funcionales
  - cod_socio->ciudad
  - Ciudad->país
- No está en 3FN
  - *país* depende transitivamente de la clave *cod_socio* a través de *ciudad*
- Solución: Descomposición en dos proyecciones
  - **SOCIO1** (cod_socio, ciudad)
  - **CIUDAD** (ciudad, país)

Forma Normal Boyce Codd(FNBC)

- Todo determinante es una clave
- Determinante: uno o varios atributos que determinan otro atributo o atributos

Cuarta Forma Normal (4FN)

- Una tabla esta en cuarta forma normal si esta en 3FN y no tiene dependencias de valores múltiples

Quinta Forma Normal (5FN)

- Elimina las anomalías que resultan de un tipo de restricción denominada *dependencias de reunión*
- Se tratan tablas que pueden dividirse en otras tablas pero luego no pueden reconstruirse
- Únicamente tiene interés a nivel teórico y su uso práctico es muy dudoso.

Forma Normal Dominio/Clave(FNDLL)

- Se basa en las definiciones de claves y dominios de atributos.
- Una relación esta en FNDLL si y sólo si cada restricción en la relación es una consecuencia de las definiciones de dominios y claves

ÍNDICES

- Los datos pueden recuperarse de una base de datos utilizando dos métodos:
  - **Método de acceso secuencial**. Se busca consecutivamente por los registros de una base de datos hasta que se cumplan ciertas coincidencias
  - **Método de acceso directo**. Si se añaden índices a la base de datos, SQL utiliza una estructura basada en árboles para recuperar la información
- Los índices se utilizan por tres razones
  - Hacer cumplir las restricciones de la integridad referencial utilizando la palabra reservada UNIQUE
  - Facilitar la ordenación de los datos basándose en los contenidos de los campos índice
  - Para optimizar la velocidad de las consultas
- La definición de una clave primaria implica la definición de un índice único que estará compuesto por dichas columnas

ÍNDICES

Creación de índices
CREATE INDEX

- Sintaxis abreviada:

CREATE INDEX nombre_indice
ON nombre_tabla (columna1, [columna2], ...);

Ejemplo:

```
SQL> CREATE INDEX ind_numerof ON facturas (numerof);

Index created.
```

ÍNDICES: Ejemplo

```
SQL> CREATE TABLE facturas(
         concepto VARCHAR2(20),
        importe NUMBER,
        numeroF NUMBER
        );

Table created.

INSERT INTO facturas VALUES('Teléfónica',125,111);

INSERT INTO facturas VALUES('Amena',25, 109);

INSERT INTO facturas VALUES('Airtel' ,135, 101);

INSERT INTO facturas VALUES('Telecable' ,195, 99);


SQL> select * from facturas;
CONCEPTO               IMPORTE   NUMEROF
-------------------- --------- ---------
Teléfónica                 125       111
Amena                       25       109
Airtel                     135       101
Telecable                  195        99


SQL> CREATE INDEX ind_numerof ON facturas (numerof);


Index created.
```

ÍNDICES

Borrado de índices
DROP INDEX

- Sintaxis abreviada:

DROP INDEX nombre_indice;

Ejemplo:

```
SQL> DROP INDEX ind_numerof;


Index dropped.
```

ÍNDICES

Indexando por más de un campo

```
SQL> CREATE INDEX ind_cmp_numerof_importe
       ON facturas (numerof, importe);

Index created.
```

ÍNDICES

Indexando de forma ascendente ASC o descendente DESC

- Sintaxis abreviada:

CREATE INDEX nombre_indice
ON nombre_tabla (columna1 [ASC|DESC],
[columna2] [ASC|DESC],
, ...);

Ejemplo:

```
SQL> CREATE INDEX ind_importe
                ON facturas (importe DESC);

Index created.
```

SINÓNIMOS

Creación de sinónimos de tablas

CREATE SYNONYM

SQL> CREATE SYNONYM facturacion FOR facturas;

Synonym created.

SQL> SELECT * FROM facturacion;

CONCEPTO	IMPORTE	NUMEROF
Teléfónica	125	111
Amena	25	109
Airtel	135	101
Telecable	195	99

SINÓNIMOS
Borrado de sinónimos de tablas
DROP SYNONYM

```
SQL> DROP SYNONYM facturacion;

Synonym dropped.

SQL>
```

VISTAS
Creación de vistas

CREATE VIEW

- Las vistas pueden definirse como tablas virtuales
- Sintaxis abreviada

```
CREATE VIEW <nombre_vista>
            [(columna1,
            columna2,
             ...)]
 AS SELECT <nombre_columnas>
            FROM <nombre_tabla>
```

- Ejemplo:

```
SQL> CREATE VIEW deudas AS SELECT * FROM facturas;

View created.

SQL> select * from deudas;
CONCEPTO                  IMPORTE    NUMEROF
-------------------- --------- ---------
Teléfónica                   125       111
Amena                         25       109
Airtel                       135       101
Telecable                    195        99
```

VISTAS
Borrado de vistas

DROP VIEW

```
SQL> DROP VIEW deudas;

View dropped.

SQL>
```

Tema 3º: DCL
Lenguaje de control de datos

- **Seguridad en la base de datos**
  - **Usuarios**
  - **Roles**
  - **Privilegios**

Seguridad en la base de datos

Creación de usuarios

CREATE USER

- **Sintaxis abreviada**

CREATE USER nombre_usuario **IDENTIFIED BY** password ;

- **Ejemplo**

```
SQL> CREATE USER juan IDENTIFIED BY cueva;

User created.

SQL> SELECT * from all_users;

USERNAME                          USER_ID CREATED
------------------------------ --------- --------
SYS                                     0 05/06/01
SYSTEM                                  5 05/06/01
OUTLN                                  11 05/06/01
DBSNMP                                 18 05/06/01
ORDSYS                                 20 05/06/01
ORDPLUGINS                             21 05/06/01
MDSYS                                  22 05/06/01
CTXSYS                                 25 05/06/01
AURORA$ORB$UNAUTHENTICATED             31 05/06/01
PEPE                                   32 07/06/01
JUAN                                   35 12/06/01
GUILLERMO                              36 12/06/01
ANTONIO                                37 12/06/01

13 rows selected.
```

Seguridad en la base de datos

Modificación de usuarios
ALTER USER

- **Sintaxis abreviada**

ALTER USER nombre_usuario **IDENTIFIED BY** password ;

- **Ejemplo**

```
SQL> ALTER USER juan IDENTIFIED BY lovelle;

User altered.

SQL>
```

Seguridad en la base de datos

Eliminación de usuarios
DROP USER

- **Sintaxis**

DROP USER nombre_usuario **[CASCADE];**

- **[CASCADE]**

  Con esta opción todos los elementos propiedad del nombre_usuario son eliminados

- **Ejemplo**

```
SQL> DROP USER juan CASCADE;

User dropped.

SQL>
```

Seguridad en la base de datos

Otorgar roles con GRANT

• Un rol es un privilegio o conjunto de privilegios que permiten a un usuario realizar ciertas funciones en la base de datos.
• Para otorgar un rol a un usuario deben utilizarse la instrucción SQL GRANT

•**Sintaxis**

GRANT tipo_rol TO nombre_usuario
[WITH ADMIN OPTION];

• **[WITH ADMIN OPTION]**
Si se pone esta opción el usuario puede otorgar roles a otros usuarios.

• Oracle permite tres tipos de roles por defecto. También se pueden crear los propios roles.
- Connect
- Resource
- DBA (DataBase Administrator)

• **Ejemplo**

```
SQL> GRANT DBA TO juan WITH ADMIN OPTION ;

Grant succeeded.
```

Seguridad en la base de datos

Roles

CONNECT, RESOURCE, DBA

- **CONNECT**
  - Este rol permite al usuario realizar consultas (SELECT), insertar (INSERT), actualizar (UPDATE) y eliminar registros de las tablas pertenecientes a otros usuarios (después de que le otorguen los permisos apropiados)
  - El usuario puede crear tablas, vistas y sinónimos.
- **Ejemplo**

```
SQL> GRANT CONNECT TO juan;
Grant succeeded.
```

- **RESOURCE**
  - **Tiene más nivel de acceso que CONNECT**
  - Este rol permite además al usuario crear procedimientos, triggers e índices.
- **Ejemplo**

```
SQL> GRANT RESOURCE TO juan;
Grant succeeded.
```

- **DBA**
  - Este rol permite al usuario todos los privilegios
- **Ejemplo**

```
SQL> GRANT DBA TO juan;
Grant succeeded.
```

Seguridad en la base de datos

Revocar roles
REVOKE

- **Eliminando roles con REVOKE**
- **Sintaxis**

REVOKE rol TO nombre_usuario ;

- **Ejemplo**

```
SQL> REVOKE CONNECT FROM juan;

Revoke succeeded.

SQL> REVOKE RESOURCE FROM juan;

Revoke succeeded.

SQL> REVOKE DBA FROM juan;

Revoke succeeded.
```

Seguridad en la base de datos

Privilegios de usuario

- Los privilegios son los permisos de los usuarios sobre los distintos elementos de la base de datos
- Los tipos de privilegios varían en función del rol del usuario
- Si un usuario crea un elemento de la base de datos puede otorgar privilegios sobre dicho elemento a otros usuarios
- Oracle define dos tipos de privilegios que pueden otorgarse a los usuarios
    - Privilegios del sistema (ver tabla 1)
    - Privilegios de los elementos de la base de datos (ver tabla 2)
- **Sintaxis**

```
GRANT privilegio_sistema TO {nombre_usuario | rol |
PUBLIC}                [WITH ADMIN OPTION];
```

- **[WITH ADMIN OPTION]**
    Si se pone esta opción el usuario puede otorgar privilegios a otros.
- **PUBLIC**
    Si se pone esta opción se otorga el privilegio a cualquiera

- **Ejemplo**

```
SQL> GRANT CREATE VIEW TO PUBLIC;

Grant succeeded.
```

Privilegios del sistema (tabla 1)	Operaciones permitidas
ALTER ANY INDEX	Allows the grantees to alter any index in any schema.
ALTER ANY PROCEDURE	Allows the grantees to alter any stored procedure, function, or package in any schema.
ALTER ANY ROLE	Allows the grantees to alter any role in the database.
ALTER ANY TABLE	Allows the grantees to alter any table or view in the schema.
ALTER ANY TRIGGER	Allows the grantees to enable, disable, or compile any database trigger in any schema.
ALTER DATABASE	Allows the grantees to alter the database.
ALTER USER	Allows the grantees to alter any user. This privilege authorizes the grantee to change another user's password or authentication method, assign quotas on any tablespace, set default and temporary tablespaces, and assign a profile and default roles.
CREATE ANY INDEX	Allows the grantees to create an index on any table in any schema.
CREATE ANY PROCEDURE	Allows the grantees to create stored procedures, functions, and packages in any schema.
CREATE ANY TABLE	Allows the grantees to create tables in any schema. The owner of the schema containing the table must have space quota on the tablespace to contain the table.
CREATE ANY TRIGGER	Allows the grantees to create a database trigger in any schema associated with a table in any schema.
CREATE ANY VIEW	Allows the grantees to create views in any schema.
CREATE PROCEDURE	Allows the grantees to create stored procedures, functions, and packages in their own schema.
CREATE PROFILE	Allows the grantees to create profiles.
CREATE ROLE	Allows the grantees to create roles.
CREATE SYNONYM	Allows the grantees to create synonyms in their own schemas.
CREATE TABLE	Allows the grantees to create tables in their own schemas. To create a table, the grantees must also have space quota on the tablespace to contain the table.
CREATE TRIGGER	Allows the grantees to create a database trigger in their own schemas.
CREATE USER	Allows the grantees to create users. This privilege also allows the creator to assign quotas on any tablespace, set default and temporary tablespaces, and assign a profile as part of a CREATE USER statement.
CREATE VIEW	Allows the grantees to create views in their own schemas.
DELETE ANY TABLE	Allows the grantees to delete rows from tables or views in any schema or truncate tables in any schema.
DROP ANY INDEX	Allows the grantees to drop indexes in any schema.
DROP ANY PROCEDURE	Allows the grantees to drop stored procedures, functions, or packages in any schema.
DROP ANY ROLE	Allows the grantees to drop roles.
DROP ANY SYNONYM	Allows the grantees to drop private synonyms in any schema.
DROP ANY TABLE	Allows the grantees to drop tables in any schema.
DROP ANY TRIGGER	Allows the grantees to drop database triggers in any schema.
DROP ANY VIEW	Allows the grantees to drop views in any schema.
DROP USER	Allows the grantees to drop users.
EXECUTE ANY PROCEDURE	Allows the grantees to execute procedures or functions (standalone or packaged) or reference public package variables in any schema.
GRANT ANY PRIVILEGE	Allows the grantees to grant any system privilege.
GRANT ANY ROLE	Allows the grantees to grant any role in the database.
INSERT ANY TABLE	Allows the grantees to insert rows into tables and views in any schema.
LOCK ANY TABLE	Allows the grantees to lock tables and views in any schema.
SELECT ANY SEQUENCE	Allows the grantees to reference sequences in any schema.
SELECT ANY TABLE	Allows the grantees to query tables, views, or snapshots in any schema.
UPDATE ANY ROWS	Allows the grantees to update rows in tables.

Seguridad en la base de datos

Otorgando privilegios de los elementos de la base de datos

Sintaxis

```
GRANT {elemento_bd | ALL [PRIVILEGES]}
            [ (columna [, column]...) ]
           [, {elemento_bd| ALL [PRIVILEGES]}
           [ (columna [, columna] ...) ] ] ...
      ON elemento_bd TO {usuario | role | PUBLIC}
            [, {usuario | role | PUBLIC}] ...
           [WITH GRANT OPTION]
```

- **[WITH GRANT OPTION]**
  Si se pone esta opción el usuario puede pasar los privilegios a otros.
- **Ejemplo**

```
SQL> connect INTERNAL/oracle
SQL> CREATE TABLE nomina (
      dni VARCHAR2(9) NOT NULL,
      salario NUMBER);
SQL> INSERT INTO nomina VALUES('12345678A',1000);
SQL> CREATE USER Guillermo IDENTIFIED BY Guillermo1;
SQL> CREATE USER Antonio IDENTIFIED BY Antonio1;
SQL> GRANT CONNECT TO Guillermo;
SQL> GRANT RESOURCE TO Antonio;
SQL> SELECT * FROM nomina;
SQL> GRANT SELECT ON nomina TO Guillermo;
SQL> GRANT SELECT, UPDATE(salario) ON nomina TO Antonio;
SQL>GRANT SELECT, UPDATE(salario) ON nomina TO Guillermo WITH
GRANT OPTION;
SQL> connect Guillermo/Guillermo1
SQL> select * from sys.nomina;

DNI         SALARIO
--------- ---------
12345678A      1000
```

Elementos de la base de datos a los que se pueden otorgar privilegios (tabla 2)

ALL
ALTER
DELETE
EXECUTE
INDEX
INSERT
REFERENCES
SELECT
UPDATE

Seguridad en la base de datos

Revocando privilegios de los elementos de la base de datos

Sintaxis

```
REVOKE {elemento_bd | ALL [PRIVILEGES]}
        [, {elemento_bd | ALL [PRIVILEGES]} ]
        ON elemento_bd FROM {usuario | role | PUBLIC}
        [, {usuario | role | PUBLIC}]
        [CASCADE CONSTRAINTS]
```

•Ejemplo

```
SQL> REVOKE select ON nomina FROM Guillermo;

Revoke succeeded.

SQL> REVOKE select ON nomina FROM Antonio;

Revoke succeeded.

SQL>
```

Tema 4º: DML
Lenguaje de manipulación de datos

- **Manipulación de datos**
  - INSERT
  - UPDATE
  - DELETE
- **Consultas**
  - SELECT

Manipulación de datos

Introducción de filas a una tabla (I)

INSERT INTO ... VALUES

- **Sintaxis abreviada**

INSERT INTO nombre_tabla
(columna1, columna2,...)
VALUES (valor1, valor2,...);

Ejemplo

```
SQL> CREATE TABLE libros (
        isbn VARCHAR2(13),
        precio NUMBER(8,2),
        titulo VARCHAR2(40));
Table created.

SQL> INSERT INTO libros
         (isbn, precio, titulo)
     VALUES('84-283-2475-1',5500,'Manual SQL');

1 row created.

SQL> SELECT * from libros;

ISBN             PRECIO TITULO
------------- --------- ----------------------------------------
84-283-2475-1       5500 Manual SQL

SQL>
```

Manipulación de datos

Introducción de filas a una tabla (II)

INSERT INTO ... VALUES

Ejemplo: Forma abreviada

```
SQL> INSERT INTO libros
     VALUES('84-481-1469-8',6700,'ORACLE8');

1 row created.

SQL> SELECT * from libros;

ISBN             PRECIO TITULO
------------- --------- ----------------------------------------
84-283-2475-1       5500 Manual SQL
84-481-1469-8       6700 ORACLE8

SQL>
```

Manipulación de datos

Modificación de filas (I)

UPDATE

- **Sintaxis abreviada**

```
UPDATE nombre_tabla SET
                    columna1 = expresión1
                    [, columna2=expresión2 ] [, ...]
                    [WHERE  condición] ;
```

Ejemplo

```
SQL> SELECT * from libros;

ISBN             PRECIO TITULO
------------- --------- ----------------------------------------
84-283-2475-1       5500 Manual SQL
84-481-1469-8       6700 ORACLE8

SQL> UPDATE libros SET precio=precio * 1.10;

2 rows updated.

SQL> select * from libros;

ISBN             PRECIO TITULO
------------- --------- ----------------------------------------
84-283-2475-1       6050 Manual SQL
84-481-1469-8       7370 ORACLE8
```

Manipulación de datos

Modificación de filas (II)

UPDATE

- **Sintaxis abreviada**

```
UPDATE nombre_tabla SET
                    columna1 = expresión1
                    [, columna2=expresión2 ] [, ...]
                    [WHERE  condición] ;
```

Ejemplo:

```
SQL> select * from libros;

ISBN             PRECIO TITULO
------------- --------- ----------------------------------------
84-283-2475-1      6050 Manual SQL
84-481-1469-8      7370 ORACLE8

SQL> UPDATE libros
     SET titulo = 'ORACLE8 programación PL/SQL'
     WHERE titulo= 'ORACLE8';

1 row updated.

SQL> select * from libros;

ISBN             PRECIO TITULO
------------- --------- ----------------------------------------
84-283-2475-1      6050 Manual SQL
84-481-1469-8      7370 ORACLE8 programación PL/SQL
```

Manipulación de datos

Borrado de filas (I)

DELETE

- **Sintaxis abreviada**

DELETE [FROM] nombre_tabla [WHERE condición] ;

Ejemplo:

```
SQL> select * from libros;

ISBN             PRECIO TITULO
-------------- --------- ----------------------------------------
84-481-1469-8       6700 ORACLE8
84-283-2475-1       5500 Manual SQL
```

Ejemplo: BORRAR TODAS LAS FILAS

```
SQL> delete libros;

2 rows deleted.

SQL> select * from libros;

no rows selected
```

Manipulación de datos

Borrado de filas(II)

DELETE

- **Sintaxis abreviada**

DELETE [FROM] nombre_tabla [WHERE condición] ;

Ejemplo:

```
SQL> select * from libros;

ISBN             PRECIO TITULO
------------- --------- ----------------------------------------
84-481-1469-8       6700 ORACLE8
84-283-2475-1       5500 Manual SQL
```

Ejemplo: borrar las filas que cumplan una condición

```
SQL> delete libros where precio> 5500;

1 row deleted.

SQL> select * from libros;

ISBN             PRECIO TITULO
------------- --------- ----------------------------------------
84-283-2475-1       5500 Manual SQL
```

Manipulación de datos

Introducción de filas desde otra tabla

INSERT INTO ... SELECT

- **Sintaxis abreviada**

INSERT INTO nombre_tabla
(columna1, columna2,...)
SELECT cláusulas_Select ;

Ejemplo:

```
SQL> CREATE TABLE paises (
          prefijo NUMBER(3),
          pais VARCHAR2(20),
          pib NUMBER(15),
          habitantes NUMBER (15));
SQL> INSERT INTO paises
          VALUES (34,'España' ,100,1000 );
SQL> INSERT INTO paises
          VALUES (51,'Perú',10,2000 );
SQL> SELECT * from paises;
   PREFIJO PAIS                          PIB HABITANTES
---------- -------------------- ---------- ----------
        34 España                       100       1000
        51 Perú                          10       2000
SQL> CREATE TABLE telefonos(
          prefijo NUMBER(3),
          pais VARCHAR2(20));
SQL> INSERT INTO telefonos (prefijo,pais)
          SELECT prefijo,pais FROM paises;
2 rows created.
SQL> select * from telefonos;
   PREFIJO PAIS
---------- --------------------
        34 España
        51 Perú
```

Consultas
SELECT

- La única forma de leer las filas contenidas en las tablas de una base de datos es mediante la instrucción SELECT
- La instrucción SELECT lee una o varias tablas y el resultado es una tabla derivada compuesta por un número determinado de columnas.
- El resultado de una SELECT es el producto cartesiano de las tablas contenidas en la cláusula FROM
- En el caso de existir cláusula WHERE se eliminarían, de la tabla resultado de la SELECT, todas aquellas filas que no cumpliesen las condiciones especificadas en dicha cláusula
- De todas las filas resultado también se eliminarán las columnas que no hayan sido indicadas en la selección <lista_columnas>

Consultas

Sintaxis abreviada de SELECT

```
SELECT  [DISTINCT | ALL] lista_columnas FROM lista_tablas

    [WHERE condición ]

    [GROUP BY expr [, expr] ...

    [HAVING condición] ]
        [{UNION | UNION ALL | INTERSECT | MINUS}
        SELECT orden ]

   [ORDER BY columna  [ASC | DESC]
        [, columna [ASC | DESC]] ...]
```

- **ALL** indica que se seleccionan todas las filas, sin eliminar filas duplicadas si las hubiera. Es el valor por defecto
- **DISTINCT** indica que se eliminarán filas duplicadas

Consultas
Ejemplos de SELECT

- **Manejando nombres de tablas en el Diccionario de datos**

```
SQL> SELECT * FROM user_tables;

SQL> describe user_tables;

SQL> SELECT table_name FROM user_tables;

SQL> -- Tablas que no lleven $ en el nombre
SQL> SELECT table_name
        FROM user_tables
        WHERE table_name NOT LIKE '%$%';

SQL> -- Tablas que empiezan por T
SQL>SELECT table_name
        FROM user_tables
        WHERE table_name LIKE 'T%';
```

Consultas

Consultas ordenadas (I)

SELECT ... ORDER BY

```
SQL> CREATE TABLE cheques (
                    codigo NUMBER,
                    pagado_a VARCHAR2(20),
                    valor NUMBER,
                    motivo VARCHAR2(20));
SQL> INSERT INTO cheques
          VALUES (34,'Juan Manuel',1000,'Viaje Córdoba');
SQL> INSERT INTO cheques
          VALUES (10,'José Manuel',2000,'Viaje Madrid');
SQL> INSERT INTO cheques
          VALUES (50,'Guillermo',200,'Curso en Madrid');
SQL> SELECT * FROM cheques;
   CODIGO PAGADO_A                VALOR MOTIVO
--------- -------------------- --------- --------------------
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
       50 Guillermo                   200 Curso en Madrid
SQL> SELECT * FROM cheques ORDER BY codigo;
   CODIGO PAGADO_A                VALOR MOTIVO
--------- -------------------- --------- --------------------
       10 José Manuel                2000 Viaje Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       50 Guillermo                   200 Curso en Madrid
SQL> SELECT * FROM cheques ORDER BY valor;
   CODIGO PAGADO_A                VALOR MOTIVO
--------- -------------------- --------- --------------------
       50 Guillermo                   200 Curso en Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
SQL> SELECT * FROM cheques ORDER BY pagado_a;

 CODIGO PAGADO_A                VALOR MOTIVO
--------- -------------------- --------- --------------------
       50 Guillermo                   200 Curso en Madrid
       10 José Manuel                2000 Viaje Madrid
       34 Juan Manuel                1000 Viaje Córdoba
```

Consultas

Consultas ordenadas (II)

SELECT ... ORDER BY ...ASC | DESC

```
SQL> SELECT * FROM cheques ORDER BY valor DESC;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       10 José Manuel                2000 Viaje Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       50 Guillermo                   200 Curso en Madrid

SQL> INSERT INTO cheques
          VALUES (60,'Antonio',2000,'Hotel en Madrid');
```

- **Ordenando por más de un atributo**

```
SQL> SELECT * FROM cheques ORDER BY valor, pagado_a;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       50 Guillermo                   200 Curso en Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       60 Antonio                    2000 Hotel en Madrid
       10 José Manuel                2000 Viaje Madrid

SQL> SELECT * FROM cheques ORDER BY valor DESC, pagado_a ASC;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       60 Antonio                    2000 Hotel en Madrid
       10 José Manuel                2000 Viaje Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       50 Guillermo                   200 Curso en Madrid
```

Consultas

FUNCIONES DE AGREGACIÓN
COUNT, SUM, AVG, MIN, MAX

```
SQL> SELECT * FROM cheques;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
       50 Guillermo                   200 Curso en Madrid
       60 Antonio                    2000 Hotel en Madrid
```

- **COUNT : contador**

```
SQL> SELECT COUNT(*) FROM cheques;

 COUNT(*)
---------
        4
```

- **SUM : suma**

```
SQL> SELECT SUM(VALOR) FROM cheques;

SUM(VALOR)
----------
      5200
```

- **AVG: media**

```
SQL> SELECT AVG(VALOR) FROM cheques;

AVG(VALOR)
----------
      1300
```

- **MAX: máximo**

```
SQL> SELECT MAX(VALOR) FROM cheques;

MAX(VALOR)
----------
      2000
```

Consultas

FUNCIONES DE AGREGACIÓN Y GROUP BY (I)

```
SQL> INSERT INTO cheques
          VALUES (90,'Antonio',5000,'Hotel en Londres');

SQL> SELECT * FROM cheques;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
       50 Guillermo                   200 Curso en Madrid
       60 Antonio                    2000 Hotel en Madrid
       90 Antonio                    5000 Hotel en Londres

SQL> SELECT pagado_a, SUM(valor) FROM cheques
          GROUP BY pagado_a;

PAGADO_A             SUM(VALOR)
-------------------- ----------
Antonio                    7000
Guillermo                   200
José Manuel                2000
Juan Manuel                1000

SQL> SELECT pagado_a, SUM(valor), COUNT(pagado_a)
          FROM cheques
          GROUP BY pagado_a;

PAGADO_A             SUM(VALOR) COUNT(PAGADO_A)
-------------------- ---------- ---------------
Antonio                    7000               2
Guillermo                   200               1
José Manuel                2000               1
Juan Manuel                1000               1
```

Consultas

Consultas ordenadas (I)

SELECT ... ORDER BY

```
SQL> CREATE TABLE cheques (
                    codigo NUMBER,
                    pagado_a VARCHAR2(20),
                    valor NUMBER,
                    motivo VARCHAR2(20));
SQL> INSERT INTO cheques
          VALUES (34,'Juan Manuel',1000,'Viaje Córdoba');
SQL> INSERT INTO cheques
          VALUES (10,'José Manuel',2000,'Viaje Madrid');
SQL> INSERT INTO cheques
          VALUES (50,'Guillermo',200,'Curso en Madrid');
SQL> SELECT * FROM cheques;
   CODIGO PAGADO_A                VALOR MOTIVO
--------- -------------------- --------- --------------------
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
       50 Guillermo                   200 Curso en Madrid
SQL> SELECT * FROM cheques ORDER BY codigo;
   CODIGO PAGADO_A                VALOR MOTIVO
--------- -------------------- --------- --------------------
       10 José Manuel                2000 Viaje Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       50 Guillermo                   200 Curso en Madrid
SQL> SELECT * FROM cheques ORDER BY valor;
   CODIGO PAGADO_A                VALOR MOTIVO
--------- -------------------- --------- --------------------
       50 Guillermo                   200 Curso en Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
SQL> SELECT * FROM cheques ORDER BY pagado_a;

 CODIGO PAGADO_A                VALOR MOTIVO
--------- -------------------- --------- --------------------
       50 Guillermo                   200 Curso en Madrid
       10 José Manuel                2000 Viaje Madrid
       34 Juan Manuel                1000 Viaje Córdoba
```

Consultas

Consultas ordenadas (II)

SELECT ... ORDER BY ...ASC | DESC

```
SQL> SELECT * FROM cheques ORDER BY valor DESC;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       10 José Manuel                2000 Viaje Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       50 Guillermo                   200 Curso en Madrid

SQL> INSERT INTO cheques
          VALUES (60,'Antonio',2000,'Hotel en Madrid');
```

- **Ordenando por más de un atributo**

```
SQL> SELECT * FROM cheques ORDER BY valor, pagado_a;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       50 Guillermo                   200 Curso en Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       60 Antonio                    2000 Hotel en Madrid
       10 José Manuel                2000 Viaje Madrid

SQL> SELECT * FROM cheques ORDER BY valor DESC, pagado_a ASC;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       60 Antonio                    2000 Hotel en Madrid
       10 José Manuel                2000 Viaje Madrid
       34 Juan Manuel                1000 Viaje Córdoba
       50 Guillermo                   200 Curso en Madrid
```

Consultas

FUNCIONES DE AGREGACIÓN
COUNT, SUM, AVG, MIN, MAX

```
SQL> SELECT * FROM cheques;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
       50 Guillermo                   200 Curso en Madrid
       60 Antonio                    2000 Hotel en Madrid
```

- **COUNT : contador**

```
SQL> SELECT COUNT(*) FROM cheques;

 COUNT(*)
---------
        4
```

- **SUM : suma**

```
SQL> SELECT SUM(VALOR) FROM cheques;

SUM(VALOR)
----------
      5200
```

- **AVG: media**

```
SQL> SELECT AVG(VALOR) FROM cheques;

AVG(VALOR)
----------
      1300
```

- **MAX: máximo**

```
SQL> SELECT MAX(VALOR) FROM cheques;

MAX(VALOR)
----------
      2000
```

Consultas

FUNCIONES DE AGREGACIÓN Y GROUP BY (I)

```
SQL> INSERT INTO cheques
          VALUES (90,'Antonio',5000,'Hotel en Londres');

SQL> SELECT * FROM cheques;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
       50 Guillermo                   200 Curso en Madrid
       60 Antonio                    2000 Hotel en Madrid
       90 Antonio                    5000 Hotel en Londres

SQL> SELECT pagado_a, SUM(valor) FROM cheques
          GROUP BY pagado_a;

PAGADO_A             SUM(VALOR)
-------------------- ----------
Antonio                    7000
Guillermo                   200
José Manuel                2000
Juan Manuel                1000

SQL> SELECT pagado_a, SUM(valor), COUNT(pagado_a)
          FROM cheques
          GROUP BY pagado_a;

PAGADO_A             SUM(VALOR) COUNT(PAGADO_A)
-------------------- ---------- ---------------
Antonio                    7000               2
Guillermo                   200               1
José Manuel                2000               1
Juan Manuel                1000               1
```

Consultas

FUNCIONES DE AGREGACIÓN Y GROUP BY (II)

```
SQL>INSERT INTO cheques
          VALUES (95,'Paloma',7000,'Hotel en Londres');

SQL> SELECT * FROM cheques;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
       50 Guillermo                   200 Curso en Madrid
       60 Antonio                    2000 Hotel en Madrid
       90 Antonio                    5000 Hotel en Londres
       95 Paloma                     7000 Hotel en Londres

SQL> SELECT motivo, MIN(valor), MAX(valor)
          FROM cheques
          GROUP BY motivo;

MOTIVO               MIN(VALOR) MAX(VALOR)
-------------------- ---------- ----------
Curso en Madrid             200        200
Hotel en Londres           5000       7000
Hotel en Madrid            2000       2000
Viaje Córdoba              1000       1000
Viaje Madrid               2000       2000
```

Consultas
SELECT...HAVING (I)

```
SQL> CREATE TABLE personal (
                    nombre VARCHAR2(10),
                    departamento VARCHAR2(10),
                    salario NUMBER,
                    diasbaja NUMBER,
                    faltas NUMBER);
SQL> INSERT INTO personal
          VALUES('Paloma','VENTAS', 1000, 10, 20);

SQL> INSERT INTO personal
          VALUES('Antonio','I+D', 5000, 1, 10);

SQL> select * from personal;

NOMBRE     DEPARTAMEN    SALARIO  DIASBAJA    FALTAS
---------- ---------- ---------- --------- ---------
Paloma     VENTAS           1000        10        20
Antonio    I+D              5000         1        10


SQL> SELECT  departamento, AVG(salario)
          FROM personal
          WHERE AVG(salario) < 3800
          GROUP BY departamento;

 WHERE AVG(salario) < 3800
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

- **WHERE no permite funciones de agregación. Para realizar esta consulta es necesario usar HAVING**

Consultas
SELECT...HAVING (II)

```
SQL> SELECT departamento, AVG (salario)
          FROM personal
          GROUP BY departamento
          HAVING AVG(salario) <3000;


DEPARTAMEN AVG(SALARIO)
---------- ------------
VENTAS             1000

SQL>SELECT departamento, AVG (salario)
          FROM personal
          GROUP BY departamento
          HAVING AVG(salario) <3000
                 AND AVG(diasbaja) <20;


DEPARTAMEN AVG(SALARIO)
---------- ------------
VENTAS             1000
```

Consultas
SELECT...HAVING (III)
COUNT, IN

```
SQL> INSERT INTO personal
          VALUES('Guillermo','VENTAS', 5000, 1, 10);

SQL> select * from personal;

NOMBRE     DEPARTAMEN   SALARIO  DIASBAJA    FALTAS
---------- ---------- --------- --------- ---------
Paloma     VENTAS          1000        10        20
Antonio    I+D             5000         1        10
Guillermo  VENTAS          5000         1        10

SQL> SELECT departamento, AVG (salario), AVG (diasbaja)
          FROM personal
          GROUP BY departamento
          HAVING COUNT(departamento)>1;

DEPARTAMEN AVG(SALARIO) AVG(DIASBAJA)
---------- ------------ -------------
VENTAS             3000           5,5

SQL> SELECT departamento, AVG (salario), AVG (diasbaja)
          FROM personal
          GROUP BY departamento
          HAVING departamento IN ('VENTAS','I+D');

DEPARTAMEN AVG(SALARIO) AVG(DIASBAJA)
---------- ------------ -------------
I+D                5000             1
VENTAS             3000           5,5
```

Consultas

SELECT con varias tablas

```
SQL> select * from personal;

NOMBRE     DEPARTAMEN    SALARIO  DIASBAJA     FALTAS
---------- ---------- ---------- --------- ---------
Paloma     VENTAS           1000        10         20
Antonio    I+D              5000         1         10
Guillermo  VENTAS           5000         1         10

SQL> select * from cheques;

   CODIGO PAGADO_A                    VALOR MOTIVO
--------- -------------------- --------- --------------------
       34 Juan Manuel                1000 Viaje Córdoba
       10 José Manuel                2000 Viaje Madrid
       50 Guillermo                   200 Curso en Madrid
       60 Antonio                    2000 Hotel en Madrid
       90 Antonio                    5000 Hotel en Londres
       95 Paloma                     7000 Hotel en Londres

SQL> select * from personal, cheques;
NOMBRE     DEPARTAMEN    SALARIO  DIASBAJA     FALTAS     CODIGO PAGADO_A                VALOR
---------- ---------- ---------- --------- ---------- --------- -------------------- ---------
MOTIVO
--------------------
Paloma     VENTAS           1000        10         20        34 Juan Manuel              1000
Viaje Córdoba

Antonio    I+D              5000         1         10        34 Juan Manuel              1000
Viaje Córdoba

Guillermo  VENTAS           5000         1         10        34 Juan Manuel              1000
Viaje Córdoba

Paloma     VENTAS           1000        10         20        10 José Manuel              2000
Viaje Madrid

. . . (continua)

18 rows selected.
```

PRODUCTO CARTESIANO 3 x 6 = 18 filas

Consultas

SELECT con varias tablas
WHERE, ORDER BY

```
SQL> SELECT p.nombre,p.departamento, c.valor
          FROM personal p, cheques c
          WHERE p.nombre=c.pagado_a;

NOMBRE     DEPARTAMEN     VALOR
---------- ---------- ---------
Antonio    I+D             2000
Antonio    I+D             5000
Guillermo  VENTAS           200
Paloma     VENTAS          7000


SQL> SELECT p.nombre,p.departamento, c.valor
          FROM personal p, cheques c
          WHERE p.nombre=c.pagado_a
          ORDER BY c.valor;

NOMBRE     DEPARTAMEN     VALOR
---------- ---------- ---------
Guillermo  VENTAS           200
Antonio    I+D             2000
Antonio    I+D             5000
Paloma     VENTAS          7000
```

Subconsultas

El resultado de una consulta se utiliza en otra consulta

- **Sintaxis**

```
SELECT * FROM tabla_1
    WHERE tabla_1.algunaColumna =
  (SELECT algunaColumna FROM tabla_2
        WHERE algunaColumna = algunValor) ;
```

- **Ejemplo**

```
CREATE TABLE piezas (
        cod NUMBER(4),
        nombre VARCHAR2(10),
         precio NUMBER);
INSERT INTO piezas VALUES(123, 'Rueda bici',100);
INSERT INTO piezas VALUES(222, 'Pedal',120);
CREATE TABLE pedidos ( fecha DATE,
        descripcion VARCHAR2(10),
         numero NUMBER(4),
        cantidad NUMBER(4)
        comentarios VARCHAR2(10));
INSERT INTO pedidos
        VALUES('01/01/01', 'Pedal',123,2, 'Pagado');
INSERT INTO pedidos
        VALUES('01/01/01', 'Rueda bici',222,2, 'Pagado');

SELECT * FROM pedidos WHERE numero=
         (SELECT cod FROM piezas WHERE nombre LIKE 'Rueda%');
```

Tema 5º: Transacciones

- Transacciones
- Comenzando una transacción
  - SET TRANSACTION READ ONLY
- Finalizando una transacción
  - COMMIT
- Interrumpiendo una transacción
  - ROLLBACK
- Puntos de salvaguarda
  - SAVEPOINT

Transacciones

- Una transacción es el conjunto de operaciones relativas a recuperación y actualización sobre la base de datos que se realizan de forma unitaria e indivisible
- Una transacción se realiza totalmente o no se realiza
- Es el modo de agrupar varias instrucciones del DML lenguaje de manipulación de datos (INSERT, DELETE y UPDATE) y asurar que dicho grupo se ejecute completamente

COMENZANDO UNA TRANSACCIÓN SÓLO LECTURA (READ ONLY)

- **Sintaxis**

SET TRANSACTION READ ONLY ;

- **Ejemplo**

```
SET TRANSACTION READ ONLY;
SELECT * FROM  personal
        WHERE nombre = 'Antonio';

 ---Otras operaciones ---
COMMIT;
```

- **La opción SET TRANSACTION READ ONLY bloquea un conjunto de registros hasta que la transacción finaliza**
- **Se puede usar la opción READ ONLY con:**
  - **SELECT**
  - **LOCK TABLE**
  - **SET ROLE**
  - **ALTER SESSION**
  - **ALTER SYSTEM**

COMENZANDO UNA TRANSACCIÓN

- **En Oracle**
  - Una transacción comienza con la primera orden SQL emitida después de terminar la transacción anterior
  - Una transacción comienza con la primera orden SQL después de conectarse a la base de datos

- **Ejemplo**

```
-- INICIO transacción

INSERT INTO personal
          VALUES('Garcilaso','VENTAS', 3000, 1, 10);
INSERT INTO personal
          VALUES('Garcia','I+D', 1000, 2, 20);

-- FIN transacción
COMMIT;
-- INICIO  nueva transacción
```

FINALIZANDO UNA TRANSACCIÓN

Finaliza la transacción haciendo definitivos los cambios realizados por el proceso

COMMIT

- **Sintaxis**

COMMIT [WORK] ;

- WORK es opcional y sólo tiene como objeto facilitar la comprensión de la orden

- **Ejemplo**

```
INSERT INTO personal
          VALUES('Garcilaso','VENTAS', 3000, 1, 10);
INSERT INTO personal
          VALUES('Garcia','I+D', 1000, 2, 20);

-- FIN transacción
COMMIT;
SELECT * FROM personal;
```

- **AUTOCOMMIT**

SET AUTOCOMMIT [ON | OFF]

- **ON Todas las operaciones que se ejecutan una a una.**
  - **Tras cada instrucción se ejecuta un COMMIT automáticamente realizándose las actualizaciones en la base de datos.**
  - **Valor por defecto en SQL*PLUS interactivo**
- **OFF Las operaciones se realizaran todas juntas cuando se ejecute explícitamente un COMMIT.**
  - **Valor por defecto en los bloques PL/SQL, dado que SQL*PLUS no recupera el control hasta que finaliza la ejecución del bloque.**

Interrumpiendo una transacción

Finaliza la transacción deshaciendo los cambios realizados por el proceso
ROLLBACK

- **Sintaxis**

ROLLBACK [WORK];

- **Ejemplo**

```
INSERT INTO personal
          VALUES('Gao','VENTAS', 3000, 1, 10);
INSERT INTO personal
          VALUES('Gerarda','I+D', 1000, 2, 20);

-- interrumpiendo
ROLLBACK;
SELECT * FROM personal;
```

Puntos de salvaguarda

Almacena parte de la transacción, que se puede recuperar posteriormente

SAVEPOINT

- **Sintaxis**

SAVEPOINT nombre;

- **Almacena la transacción hasta ese momento**
- **Para recuperar la transacción**

ROLLBACK [WORK] TO SAVEPOINT nombre;

- **Ejemplo**

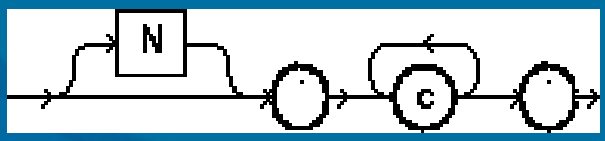
```
INSERT INTO personal
          VALUES('Gao','VENTAS', 3000, 1, 10);
SAVEPOINT A;
INSERT INTO personal
          VALUES('Gerarda','I+D', 1000, 2, 20);
SAVEPOINT B;

DELETE personal;

-- volviendo atrás
ROLLBACK TO SAVEPOINT B;
COMMIT;

SELECT * FROM personal;
```

Anexo A
Elementos básicos de SQL

- Literales o constantes
  - Cadenas de Texto
  - Enteros
  - Números
- Tipos de datos

Literales y Constantes
Cadenas de texto

Cadena_texto::=



N especifica la representación del conjunto de caracteres nacional

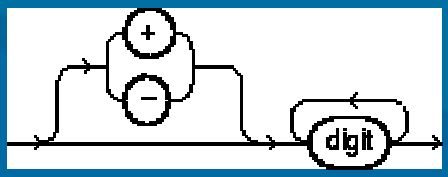
c Es cualquier carácter, excepto comilla simple (').

```
'Hola'
'Hola a todos'
'pepe'
```

Literales y Constantes

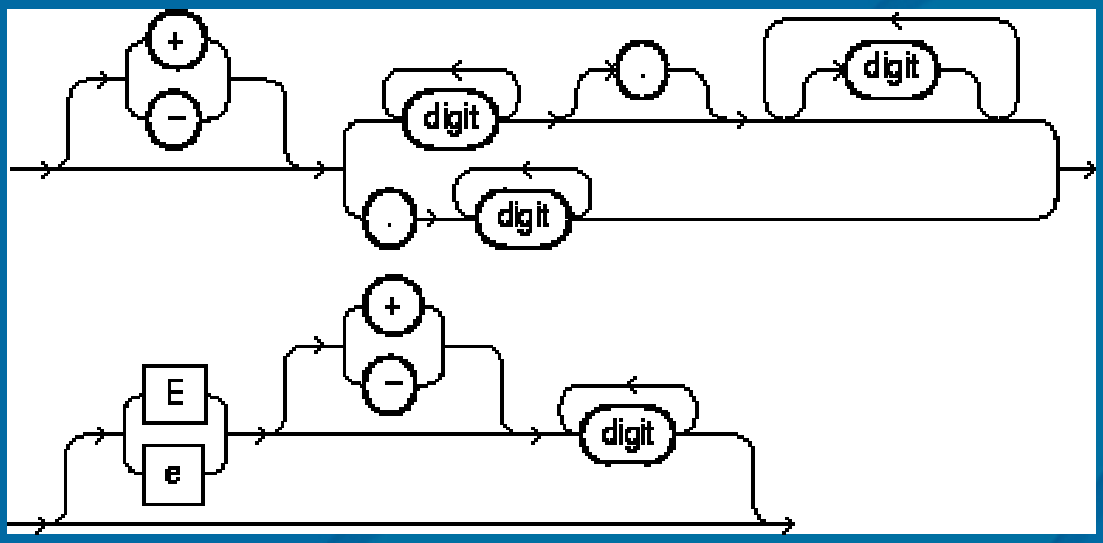
Enteros

Entero::=



Números

Número::=



Tipos de datos SQL estándar

Tipo de Datos	Longitud	Descripción
BINARY	1 byte	Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos Binario.
BIT	1 byte	Valores Si/No ó True/False
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número incrementado automáticamente (de tipo Long)
CURRENCY	8 bytes	Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	8 bytes	Un valor de fecha u hora entre los años 100 y 9999.
SINGLE	4 bytes	Un valor en punto flotante de precisión simple con un rango de $-3.402823*10^{38}$ a $-1.401298*10^{-45}$ para valores negativos, $1.401298*10^{-45}$ a $3.402823*10^{38}$ para valores positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con un rango de $-1.79769313486232*10^{308}$ a $-4.94065645841247*10^{-324}$ para valores negativos, $4.94065645841247*10^{-324}$ a $1.79769313486232*10^{308}$ para valores positivos, y 0.
SHORT	2 bytes	Un entero corto entre -32,768 y 32,767.
LONG	4 bytes	Un entero largo entre -2,147,483,648 y 2,147,483,647. En Oracle tiene otro significado
LONGTEXT	1 byte por carácter	De cero a un máximo de 1.2 gigabytes.
LONGBINARY	Según se necesite	De cero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por caracter	De cero a 255 caracteres.

La siguiente tabla recoge los sinónimos de los tipos de datos definidos

Tipo de Dato	Sinónimos
BINARY	VARBINARY
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO
BYTE	INTEGER1
COUNTER	AUTOINCREMENT
CURRENCY	MONEY
DATETIME	DATE TIME TIMESTAMP
SINGLE	FLOAT4 IEEESINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEEDOUBLE NUMBER NUMERIC
SHORT	INTEGER2 SMALLINT
LONG	INT INTEGER INTEGER4
LONGBINARY	GENERAL OLEOBJECT
LONGTEXT	LONGCHAR MEMO NOTE
TEXT	ALPHANUMERIC CHAR CHARACTER STRING VARCHAR
VARIANT (No Admitido)	VALUE

Referencias

Tutoriales en la web

- http://www.programacion.net/cursos/sql/
- http://www.aulaclic.org/sql/f_sql.htm
- http://w3.one.net/~jhoffman/sqltut.htm
- www.informit.com (libros en línea)
- www.oracle.com

Referencias

Libros

- *[Freeze W.S. 1998]* **SQL: Manual de referencia del programador.** Editorial Paraninfo
- *[Loney K.* 2000] **ORACLE 8 Manual del Administrador** Oracle Press/Osborne/McGrawHill
- *[Urman 2000] S.* **ORACLE 8 Programación PL/SQL** Oracle Press/Osborne/McGrawHill
- ***[A. de Miguel et al. 1999]* Fundamentos y modelos de Bases de Datos** Editorial Rama