

# ***Objetos de Base de Datos***

**UTN - FRBA**  
**Ing. en Sistemas de Información**  
***Gestión de Datos***

**Prof.: Ing. Juan Zaffaroni**

# Objetos

- Tablas
- Tablas Temporales
- Tablas Anidadas (Ora)
- Constraints
- Secuencias
- Views
- Snapshots /Summarized Tables /Materialized Views
- Sinónimos/NickNames
- DataBase Links (Ora)
- Directories (Ora)
- Indices
- Tablas Organizadas por Índice (Ora)
- Clusters (Ora)
- Stored Procedures
- Funciones propias del motor
- Funciones de usuario
- Packages (Ora / DB2)
- Triggers
- Esquemas (Ora)

# Tablas

Es la unidad básica de almacenamiento de datos. Los datos están almacenados en filas y columnas. Son de existencia permanente y poseen un nombre identificador único por esquema o por base de datos (dependiendo del motor de base de datos).

Cada columna tiene entre otros datos un nombre, un tipo de datos y un ancho (este puede estar predeterminado por el tipo de dato).

# Tablas

## Ejemplo ORACLE

```
CREATE TABLE ordenes  
( N_orden    NUMBER,  
  N_cliente  NUMBER,  
  F_orden    DATE,  
  C_estado  NUMBER,  
  F_alta_audit DATE,  
  D_usuario  VARCHAR2(20) );
```

## Motor SQL Server

```
CREATE TABLE ordenes (  
    N_orden INT NULL ,  
    N_cliente INT NULL ,  
    F_orden DATETIME NULL ,  
    C_estado SMALLINT NULL ,  
    F_alta_audit TIMESTAMP NULL ,  
    D_usuario VARCHAR(20) NULL )
```

# Tablas Temporales

Son tablas creadas cuyos datos son de existencia temporal.  
No son registradas en las tablas del diccionario de datos.  
No es posible alterar tablas temporarias. Si eliminarlas y crear los índices temporales que necesite una aplicación.  
Las actualizaciones a una tabla temporal podrían no generar ningún log transaccional si así se configurara.

## Tipos de Tablas

- De Sesión (locales)
- Globales

## Tipos de Creación

- Explícita
- Implícita

# Tablas Temporales

## Tipos de Tablas Temporales

### De Sesión (locales)

Son visibles sólo para sus creadores durante la misma sesión (conexión) a una instancia del motor de BD.

Las tablas temporales locales se eliminan cuando el usuario se desconecta o cuando decide eliminar la tabla durante la sesión.

### Globales

Las tablas temporales globales están visibles para cualquier usuario y sesión una vez creadas. Su eliminación depende del motor de base de datos que se utilice.

# Tablas Temporales

## Tipos de Creación

### Creación Explícita.

Este tipo de creación se realiza mediante la instrucción CREATE. De manera explícita se deberá crear la tabla indicando el nombre, sus campos, tipos de datos y restricciones.

### Creación Implícita

Se pueden crear tablas temporales a partir del resultado de una consulta SELECT.

# Tablas Temporales

## Por qué utilizarlas?

### Como almacenamiento intermedio de Consultas Muy Grandes:

Por ejemplo, se tiene una consulta SELECT que realiza **"JOINS"** con ocho tablas.

Muchas veces las consultas con varios **"JOINS"** pueden funcionar de manera poco performante.

Una técnica para intentar es la de dividir una consulta grande en consultas más pequeñas. Si usamos tablas temporales, podemos crear tablas con resultados intermedios basados en consultas de menor tamaño, en lugar de intentar ejecutar una consulta única que sea demasiado grande y múltiples **"JOINS"**.



# Tablas Temporales

## Por qué utilizarlas? (Cont.)

**Para optimizar accesos a una consulta varias veces en una aplicación:**

Por ejemplo, usted está utilizando una consulta que tarda varios segundos en ejecutarse, pero sólo muestra un conjunto acotado de resultados, el cual desea utilizar en varias áreas de su procedimiento almacenado, pero cada vez que se llama se debe volver a ejecutar la consulta general.

Para resolver esto, puede ejecutar la consulta una sola vez en el procedimiento, llenando una tabla temporal, de esta manera se puede hacer referencia a la tabla temporal en varios lugares en su código, sin incurrir en una sobrecarga de resultados adicional.

# Tablas Temporales

## Por qué utilizarlas? (Cont.)

### **Para almacenar resultados intermedios en una aplicación:**

Por ejemplo, usted está necesita en un determinado proceso generar información que se irá actualizando y/o transformando en distintos momentos de la ejecución, sin querer actualizar o impactar a tablas reales de la BD hasta el final del procedimiento.

Para resolver esto, puede crear una tabla temporal de sesión durante la ejecución del procedimiento, realizando en ella inserciones, modificaciones, borrado y/o transformación de datos. Al llegar al final del procedimiento, con los datos existentes en la tabla temporal se actualizará la o las tablas físicas de la BD que corresponda.

# Tablas Temporales

## Ejemplo SQLServer – Tabla de Sesión

### Creación Explícita

```
CREATE TABLE #ordenes_pendientes (  
    N_orden INTEGER,  
    N_cliente INTEGER,  
    F_orden DATE,  
    I_Total DECIMAL(15 , 2),  
    C_estado SMALLINT,  
    F_alta_audit TIMESTAMP,  
    D_usuario VARCHAR(20) )  
WITH NO LOG;
```

```
INSERT INTO #ordenes_Pendientes  
SELECT * FROM ordenes WHERE c_estado = 1
```

### Creación Implícita

```
SELECT *  
INTO #ordenes_Pendientes  
FROM ordenes  
WHERE c_estado = 1
```

Tanto el ejemplo de creación explícita y el de implícita generarán la misma tabla temporal con los mismos datos.

# Tablas Temporales

## Ejemplo Informix – Tabla de Sesión

### Creación Explícita

```
CREATE TEMP TABLE ordenes_pendientes (  
    N_orden INTEGER,  
    N_cliente INTEGER,  
    F_orden DATE,  
    I_Total   DECIMAL(15 , 2),  
    C_estado SMALLINT,  
    F_alta_audit DATETIME YEAR TO SECOND,  
    D_usuario   VARCHAR(20) )  
WITH NO LOG;
```

```
INSERT INTO ordenes_Pendientes  
(SELECT * FROM ordenes WHERE c_estado = 1)
```

### Creación Implícita

```
SELECT *  
    FROM ordenes  
    WHERE c_estado = 1  
INTO TEMP ordenes_Pendientes  
WITH NO LOG;
```

Tanto el ejemplo de creación explícita y el de implícita generarán la misma tabla temporal con los mismos datos.

# Tablas Temporales

## Ejemplo DB2 – Tabla Global

### Creación Explícita

```
DECLARE GLOBAL TEMPORARY TABLE
    ordenes_pendientes
    LIKE ordenes
    ON COMMIT PRESERVE ROWS NOT
    LOGGED IN SESIONTEMP;
```

```
INSERT INTO ordenes_Pendientes
    SELECT * FROM ordenes WHERE c_estado = 1
```

### Creación Implícita

```
SELECT *
    FROM ordenes
    WHERE c_estado = 1
    INTO TEMP ordenes_Pendientes
```

Tanto el ejemplo de creación explícita y el de implícita generarán la misma tabla temporal con los mismos datos.

# Tablas Temporales

## Ejemplo ORACLE – Tabla Temporal

### Creación Explícita

```
DECLARE GLOBAL TEMPORARY TABLE
    ordenes_pendientes (
        N_orden INTEGER,
        N_cliente INTEGER,
        F_orden DATE,
        I_Total   NUMERIC(15 , 2),
        C_estado SMALLINT,
        F_alta_audit DATE,
        D_usuario  VARCHAR2(20) )
ON COMMIT DELETE ROWS;
```

```
INSERT INTO ordenes_Pendientes
SELECT * FROM ordenes WHERE c_estado = 1
```

# Tablas Anidadas

Es posible crear una tabla con una columna cuyo tipo de dato sea otra tabla. De esta forma, las tablas pueden anidarse dentro de otras tablas como valores en una columna.

```
CREATE TYPE address_t AS OBJECT (  
    street VARCHAR2(30),  
    city VARCHAR2(20),  
    state CHAR(2),  
    zip CHAR(5) );
```

```
CREATE TYPE address_tab IS TABLE OF address_t;
```

```
CREATE TABLE customers (  
    custid NUMBER,  
    address address_tab )  
NESTED TABLE address STORE AS customer_addresses;
```

```
INSERT INTO customers VALUES  
    (1, address_tab(  
        address_t('101 First', 'Redwood Shores', 'CA', '94065'),  
        address_t('123 Maple', 'Mill Valley', 'CA', '90952')  
    )  
);
```

# Constraints

## Integridad de Entidad

La integridad de entidades es usada para asegurar que los datos pertenecientes a

una misma tabla tienen una única manera de identificarse, es decir que cada fila de cada tabla tenga una primary key capaz de identificar unívocamente una fila y esa no puede ser nula

**PRIMARY KEY CONSTRAINT:** Puede estar compuesta por una o más columnas, y deberá representar unívocamente a cada fila de la tabla. No debe permitir valores nulos (depende del motor de base de datos).

## Integridad Referencial

La integridad referencial es usada para asegurar la coherencia entre datos de dos tablas.

**FOREIGN KEY CONSTRAINT:** Puede estar compuesta por una o más columnas, y estará referenciando a la PRIMARY KEY de otra tabla.

Los constraints referenciales permiten a los usuarios especificar claves primarias y foráneas para asegurar una relación PADRE-HIJO (MAESTRO-DETALLE).



# Constraints

## Tipos de Constraints Referenciales

### Ciclic Referential Constraint.

Asegura una relación de PADRE-HIJO entre tablas. Es el más común.

Ej. CLIENTE → FACTURAS

### Self Referencing Constraint.

Asegura una relación de PADRE-HIJO entre la misma tabla.

Ej. EMPLEADOS → EMPLEADOS

### Multiple Path Constraint.

Se refiere a una PRIMARY KEY que tiene múltiples FOREIGN KEYS. Este caso también es muy común.

Ej. CLIENTES → FACTURAS  
CLIENTES → RECLAMOS

# Constraints

## Integridad Semántica

La integridad semántica es la que nos asegura que los datos que vamos a almacenar tengan una apropiada configuración y que respeten las restricciones definidas sobre los dominios o sobre los atributos.

- **DATA TYPE**
- **DEFAULT**
- **UNIQUE**
- **NOT NULL**
- **CHECK**

# Constraints

## Integridad Semántica

**DATA TYPE:** Este define el tipo de valor que se puede almacenar en una columna.

**DEFAULT CONSTRAINT:** Es el valor insertado en una columna cuando al insertar un registro ningún valor fue especificado para dicha columna. El valor default por default es el NULL.

Se aplica a columnas no listadas en una sentencia INSERT.

El valor por default puede ser un valor literal o una función SQL (USER, TODAY, etc.)

Aplicado sólo durante un INSERT (NO UPDATE).

**UNIQUE CONSTRAINT:** Especifica sobre una o más columnas que la inserción o actualización de una fila contiene un valor único en esa columna o conjunto de columnas.

**NOT NULL CONSTRAINT:** Asegura que una columna contenga un valor durante una operación de INSERT o UPDATE. Se considera el NULL como la ausencia de valor.

# Constraints

## Integridad Semántica (Cont.)

**CHECK CONSTRAINT:** Especifica condiciones para la inserción o modificación en una columna. Cada fila insertada en una tabla debe cumplir con dichas condiciones.

Actúa tanto en el INSERT, como en el UPDATE.

Es una expresión que devuelve un valor booleano de TRUE o FALSE.  
Son aplicados para cada fila que es INSERTADA o MODIFICADA.

Todas las columnas a las que referencia deben ser de la misma tabla (la corriente).

No puede contener subconsultas, secuencias, funciones (de fecha, usuario) ni pseudocolumnas.

Todas las filas existentes en una tabla deben pasar un nuevo constraint creado para dicha tabla. En el caso de que alguna de las filas no cumpla, no se podrá crear dicho constraint o se creará en estado deshabilitado.

# Constraints

## Tipos de Constraints

- Existen dos métodos para definir constraints.

### Restricciones a nivel de Columna

Ej.

```
CREATE TABLE ordenes (  
    N_orden    INT PRIMARY KEY,  
    N_cliente  INT,  
    F_orden    DATE,  
    C_estado   SMALLINT,  
    F_alta_audit DATE,  
    D_usuario  VARCHAR(20) );
```

### Restricciones a nivel de Tabla

Ej.

```
CREATE TABLE items_ordenes (  
    N_orden    INT,  
    N_item      SMALLINT,  
    C_producto INT,  
    Q_cantidad INT,  
    I_precunit  NUMERIC(9,3),  
    PRIMARY KEY (n_orden, n_item) );
```

Cuando la restricción es sobre un grupo de columnas se debe utilizar una restricción a nivel de tabla, cuando es sobre sólo una columna puede utilizarse cualquiera de los dos modos.

# Constraints

## Ejemplos de PRIMARY KEY

### Restricciones a nivel de Columna

Ej.

```
CREATE TABLE ordenes (  
    N_orden    INT PRIMARY KEY,  
    N_cliente  INT,  
    F_orden    DATE,  
    C_estado   SMALLINT,  
    F_alta_audit DATE,  
    D_usuario  VARCHAR(20) );
```

### Restricciones a nivel de Tabla

Ej.

```
CREATE TABLE items_ordenes (  
    N_orden    INT,  
    N_item      SMALLINT,  
    C_producto  INT,  
    Q_cantidad  INT,  
    I_precunit  NUMERIC(9,3),  
    PRIMARY KEY (n_orden, n_item) );
```

# Constraints

## Ejemplos de FOREIGN KEY

### Restricciones a nivel de Columna

```
CREATE TABLE ordenes (  
    N_orden INTEGER PRIMARY KEY,  
    N_cliente INTEGER,  
    F_orden DATE,  
    C_estado SMALLINT,  
    F_alta_audit DATE,  
    D_usuario VARCHAR(20)  
);
```

```
CREATE TABLE items_ordenes (  
    N_orden INT REFERENCES ordenes,  
    N_item SMALLINT,  
    C_producto INT,  
    Q_cantidad INT,  
    I_precunit NUMERIC(9,3),  
    PRIMARY KEY (n_orden, n_item));
```

### Restricciones a nivel de Tabla

```
CREATE TABLE items_ordenes (  
    N_orden INT REFERENCES ordenes,  
    N_item SMALLINT,  
    C_producto INTEGER,  
    Q_cantidad INTEGER,  
    I_precunit DECIMAL ( 9,3),  
    PRIMARY KEY (n_orden, n_item) );
```

```
CREATE TABLE mov_stock (  
    N_stock INT,  
    C_movimiento SMALLINT,  
    N_orden INT,  
    N_item SMALLINT,  
    C_producto INT,  
    Q_cantidad INT,  
    FOREIGN KEY (n_orden, n_item)  
    REFERENCES items_ordenes);
```

# Constraints

## Ejemplos de SELF REFERENCING CONSTRAINT

### Restricciones a nivel de Columna

```
CREATE TABLE empleados (  
    N_empleado    INTEGER PRIMARY KEY,  
    D_Apellido    VARCHAR(60),  
    D_nombres     VARCHAR(60),  
    N_cuil        NUMERIC(11,0),  
    N_jefe        INTEGER,  
    FOREIGN KEY (n_jefe) REFERENCES empleados (n_empleado) );
```

El motor no permitirá ingresar un empleado cuyo nro. de jefe no exista como número de empleado.

Lo que si permitirá es ingresar un nro. de jefe NULO.



# Constraints

## Ejemplos de DEFAULT

```
CREATE TABLE ordenes
(
    N_orden NUMBER NOT NULL,
    N_cliente NUMBER,
    F_orden DATE,
    C_estado NUMBER DEFAULT 1,
    F_alta_audit DATE DEFAULT CURRENT_DATE,
    D_usuario VARCHAR2(20) DEFAULT USER
);
```

Al de ejecutar el siguiente comando

```
INSERT INTO ordenes (n_orden, n_cliente, f_orden) VALUES (117, 10, '12-JAN-2013')
```

El motor de base de datos insertará el siguiente registro en la tabla **ordenes**

N_orden	117
N_cliente	10
F_orden	12-JAN-2013
C_estado	1
F_alta_audit	Fecha del insert
D_usuario	ID del usuario que realizó el insert

# Constraints

## Ejemplos de NOT NULL

```
CREATE TABLE ordenes
(
    N_orden    NUMBER NOT NULL,
    N_cliente  NUMBER,
    F_orden    DATE,
    C_estado   NUMBER NOT NULL,
    F_alta_audit DATE,
    D_usuario  VARCHAR2(20)
);
```

# Constraints

## Ejemplos de UNIQUE

### Restricciones a nivel de Columna

```
CREATE TABLE empleados (  
    N_empleado NUMERIC PRIMARY KEY,  
    D_Apellido VARCHAR(60),  
    D_nombres VARCHAR(60),  
    N_cuil NUMERIC(11,0) UNIQUE,  
    F_nacimiento DATE,  
    F_ingreso DATE,  
    N_jefe NUMERIC);
```

### Restricciones a nivel de Tabla

```
CREATE TABLE empleados (  
    N_empleado NUMERIC PRIMARY KEY,  
    D_Apellido VARCHAR(60),  
    D_nombres VARCHAR(60),  
    T_docum NUMERIC(2,0),  
    N_docum NUMERIC(11,0),  
    F_nacimiento DATE,  
    F_ingreso DATE,  
    N_jefe NUMERIC,  
    UNIQUE (t_docum, n_docum) );
```

La tabla empleados tiene cómo clave primaria al atributo **n\_empleado**.  
Con una restricción de UNIQUE podemos representar claves alternas.

En el primer ejemplo, el n\_cuil es un atributo que posee valores únicos para cada fila de la tabla **empleados**.

En el segundo ejemplo, la clave compuesta por los atributos t\_docum y n\_docum (tipo y número de documento) posee valores únicos para cada fila de la tabla **empleados**.

# Constraints

## Ejemplos de CHECK

### Restricciones a nivel de Columna

```
CREATE TABLE ordenes
(  
  N_orden NUMBER NOT NULL,  
  N_cliente NUMBER,  
  F_orden DATE,  
  C_estado NUMBER CHECK (c_estado IN (1,2,3)),  
  F_alta_audit DATE,  
  D_usuario VARCHAR2(20)  
);
```

### Restricciones a nivel de Tabla

```
CREATE TABLE empleados
(  
  N_empleado NUMBER,  
  D_Apellido VARCHAR2(60),  
  D_nombres VARCHAR2(60),  
  N_cuil NUMBER (11) UNIQUE,  
  F_nacimiento DATE,  
  F_ingreso DATE,  
  N_jefe NUMBER,  
  CHECK (F_nacimiento < F_ingreso)  
);
```

# Secuencias

Los generadores de secuencias proveen una serie de números secuenciales, especialmente usados en entornos multiusuarios para generar una números secuenciales y únicos sin el overhead de I/O a disco o el lockeo transaccional.

Los motores de base de datos proveen diferentes formas de implementar secuencias a través de:

- Tipo de Dato de una columna (Informix)
- Propiedades de una columna (SqlServer, Mysql, DB2)
- Objeto Sequence (Oracle, Informix, PostgreSQL, DB2, SqlServer)

# Secuencias

- Tipo de Dato de una columna
  - Motor BD Informix SERIAL.
- Propiedades de una columna
  - Motor SqlServer IDENTITY
  - Motor Mysql AUTO\_INCREMENT
  - Motor DB2 IDENTITY
- Objeto Sequence
  - Motores Oracle, Informix, PostgreSQL, DB2, SqlServer.
  - CREATE SEQUENCE

# Secuencias

## Tipo de Dato de una columna

El motor Informix posee varios tipo de datos SERIAL, SERIAL8 y BIGSERIAL que permiten realizar lo mismo que un objeto secuencia. Al insertar una fila en dicha tabla y asignarle un valor cero, el motor va a buscar el próximo nro. del más alto existente en la tabla.

```
CREATE TABLE ordenes (  
    N_orden SERIAL,  
    N_cliente INTEGER,  
    F_orden DATE,  
    I_Total DECIMAL(15 , 2),  
    C_estado SMALLINT,  
    F_alta_audit DATETIME YEAR TO SECOND,  
    D_usuario VARCHAR(20)  
    );
```

```
INSERT INTO ordenes  
    VALUES (0, 17, '15/05/2006', 1, CURRENT, "user2")
```

# Secuencias

## Propiedades de una columna

Existen motores que poseen propiedades de columna que permite realizar lo mismo que una secuencia. Al insertar una fila en dicha tabla, el motor va a buscar el próximo nro. del más alto existente en la tabla.

Ej. SQLServer **IDENTITY**

```
CREATE TABLE ordenes (  
  N_orden int IDENTITY (1, 1),  
  N_cliente int NULL ,  
  F_orden datetime NULL ,  
  I_Total decimal(15 , 2),  
  C_estado smallint NULL ,  
  F_alta_audit datetime NULL ,  
  D_usuario varchar (20) NULL ).
```

Ej. MySQL **AUTO\_INCREMENT**

```
CREATE TABLE animales  
( animal_id INT AUTO_INCREMENT,  
  nombre CHAR(30) NOT NULL);  
  
INSERT INTO animales ('perro'), ('gato');
```



# Secuencias

## Objeto SEQUENCE

Una secuencia debe tener un nombre, debe ser ascendente o descendente, debe tener definido el intervalo entre números, tiene definidos métodos para obtener el próximo número ó el actual (entre otros).

### Ej. SEQUENCE ORACLE

```
CREATE SEQUENCE sqc_orden_nro  
INCREMENT BY 1  
START WITH 10  
MAXVALUE 9999  
NOCYCLE  
NOCACHE;
```

```
INSERT INTO ordenes  
VALUES (sqc_orden_nro.NEXTVAL, 17,  
to_date('15/05/2006', 'dd/mm/yyyy'),  
1, SYSDATE, USER);
```

```
INSERT INTO items_ordenes  
VALUES (sqc_orden_nro.CURRVAL, 1,  
176, 20, 10.57);
```

# Views

Una view es un conjunto de columnas, ya sea reales o virtuales, de una misma tabla o no, con algún filtro determinado o no.

De esta forma, es una presentación adaptada de los datos contenidos en una o más tablas, o en otras vistas. Una vista toma la salida resultante de una consulta y la trata como una tabla.

Se pueden usar vistas en la mayoría de las situaciones en las que se pueden usar tablas.

# Views

- Tiene un nombre específico
- No aloca espacio de almacenamiento
- No contiene datos almacenados.
- Está definida por una consulta que consulta datos de una o varias tablas.

# Views

Las vistas se pueden utilizar para:

- Suministrar un **nivel adicional de seguridad** restringiendo el acceso a un conjunto predeterminado de filas o columnas de una tabla.
- **Ocultar la complejidad** de los datos.
- **Simplificar sentencias** al usuario.
- Presentar los datos desde una **perspectiva diferente** a la de la tabla base.
- **Aislar a las aplicaciones** de los cambios en la tabla base.

# Views

## RESTRICCIONES:

- No se pueden crear índices en las Views
- Una view depende de las tablas a las que se haga referencia en ella, si se elimina una tabla todas las views que dependen de ella se borrarán o se pasará a estado INVALIDO, dependiendo del motor. Lo mismo para el caso de borrar una view de la cual depende otra view.
- Algunas views tienen restringido los: Inserts, Deletes, Updates.
  - Aquellas que tengan joins
  - Una función agregada
  - Trigger INSTEAD OF
- Al crear la view el usuario debe tener permiso de select sobre las columnas de las tablas involucradas.
- No es posible adicionar a una View las cláusulas de: ORDER BY y UNION.

# Views

## RESTRICCIONES:

- Tener en cuenta ciertas restricciones para el caso de Actualizaciones:
  - Si en la tabla existieran campos que no permiten nulos y en la view no aparecen, los inserts fallarían
  - Si en la view no aparece la primary key los inserts podrían fallar
  - No se pueden definir Triggers sobre una Vista.
  - Se puede borrar filas desde una view que tenga una columna virtual.
  - Con la opción WITH CHECK OPTION, se puede actualizar siempre y cuando el chequeo de la opción en el where sea verdadero.

# Views

## Ejemplo ORACLE

```
CREATE VIEW V_Ordenes_Pendientes  
    (Orden_nro, Fecha_orden, Cliente, Importe_total)  
AS  
  
SELECT n_orden, f_orden, d_apellido || ', ' || d_nombre, i_total  
    FROM ordenes, clientes  
    WHERE ordenes.n_cliente = clientes.n_cliente  
        AND c_estado = 1;
```

# Views

## Ejemplo ORACLE

```
CREATE VIEW V_clientes_california  
AS
```

```
SELECT *  
  FROM customer  
 WHERE state='CA'
```

```
WITH CHECK OPTION
```



# **Snapshots/Materialized Views /Summarized Tables**

Los snapshots, también llamados vistas materializadas o tablas sumariadas, son objetos del esquema de una BD que pueden ser usados para sumarizar, precomputar, distribuir o replicar datos. Se utilizan sobre todo en DataWarehouse, sistemas para soporte de toma de decisión, y para computación móvil y/o distribuida.

Consumen espacio de almacenamiento en disco. Deben ser recalculadas o refrescadas cuando los datos de las tablas master cambian. Pueden ser refrescadas en forma manual, o a intervalos de tiempo definidos dependiendo el motor de BD.

# Snapshots/Materialized Views /Summarized Tables

Ej. Objeto Materialized View en Oracle

```
CREATE MATERIALIZED VIEW ST_total_ordenes_clientes  
BUILD DEFERRED  
REFRESH COMPLETE  
AS SELECT A.C_cliente, B.D_apellido, B.D_nombre,  
          SUM(A.I_total) AS Total_ordenes  
FROM ordenes A, clientes B  
WHERE A.C_cliente = B.C_cliente  
GROUP BY A.C_cliente, B.D_apellido, B.D_nombre ;
```

La cláusula **BUILD DEFERRED** causa que en la creación de la vista materializada no se realice el INSERT de los datos, que se realizará en el momento que se ejecute el próximo refresco. El valor por defecto es **BUILD IMMEDIATE**, que sí inserta los datos.

La cláusula **REFRESH COMPLETE** indica que el método de refresco será completo, que ejecuta nuevamente la subconsulta. De otra forma, la opción **FAST** le indica actualizar de acuerdo a los cambios que han ocurrido en la tabla master.

# Indices

Los índices son estructuras opcionales asociadas a una tabla.

La función de los índices es la de permitir un acceso más rápido a los datos de una tabla, se pueden crear distintos tipos de índices sobre uno o más campos.

Los índices son lógicamente y físicamente independientes de los datos en la tabla asociada. Se puede crear o borrar un índice en cualquier momento sin afectar a las tablas base o a otros índices.

# Indices (Cont.)

## TIPOS DE INDICES

### **Btree Index**

Estructura de índice estándar y más utilizada.

### **Btree Cluster Index**

Este tipo de índice provoca al momento de su creación que físicamente los datos de la tabla sean ordenados por el mismo. (Informix / SQLServer / DB2)

### **Bitmap Index (Oracle)**

Son utilizados para pocas claves con muchas repeticiones

Cada bit en el Bitmap corresponde a una fila en particular.

Si el bit esta en on significa que la fila con el correspondiente rowid tiene el valor de la clave.

### **Hash Index (MySql)**

Están implementados en tablas de hash y se basan en otros indices Btree existentes para una tabla. Si una tabla entra integramente en memoria, la manera más rápida de ejecutar consultas sobre ella es usando un hash index.

# Indices (Cont.)

## TIPOS DE INDICES (Cont.)

### **Functional Index / Function based Index**

Son indices cuya clave deriva del resultado de una función.

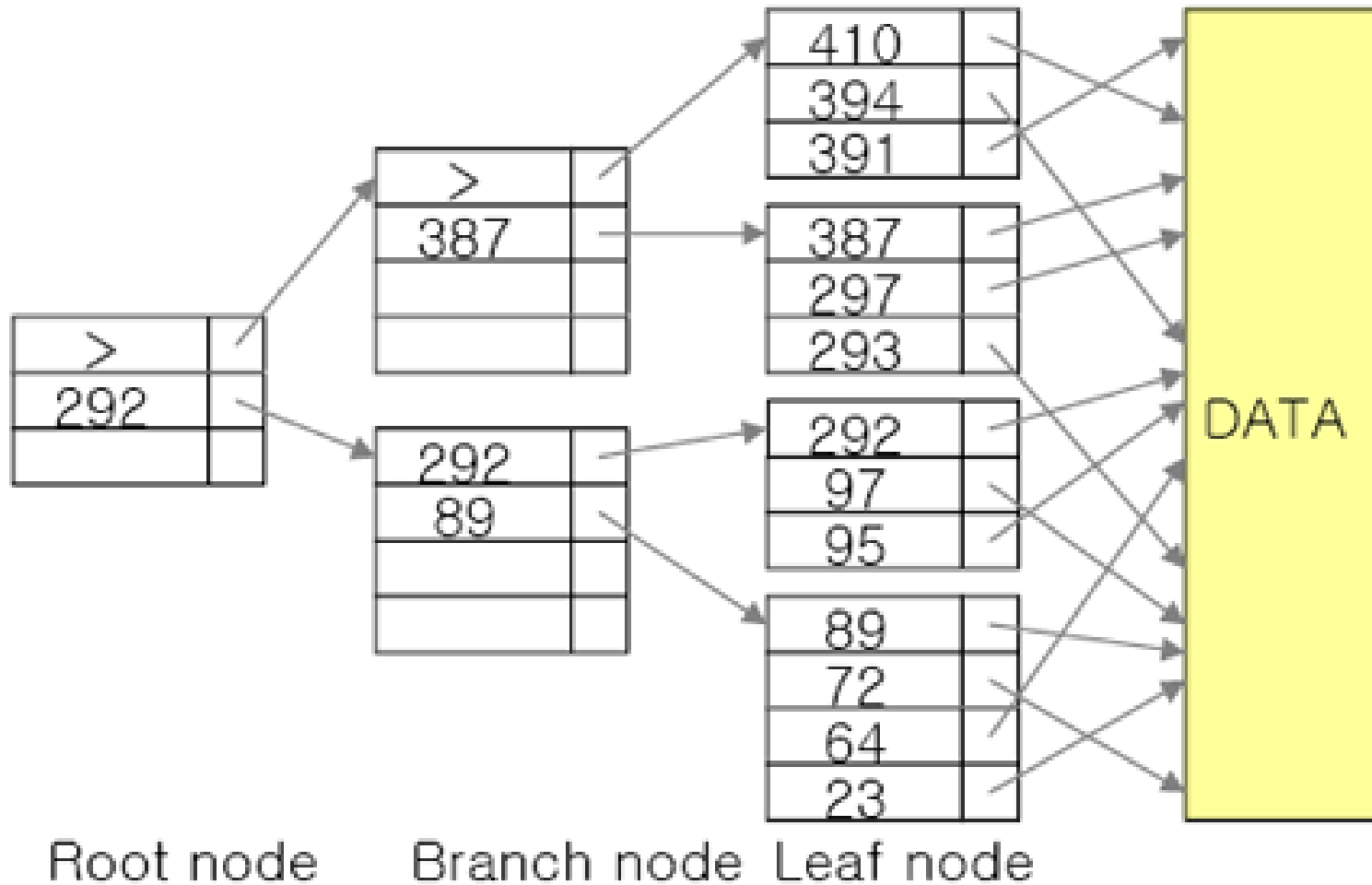
En general las funciones deben ser funciones definidas por un usuario.

### **Reverse Key Index (Oracle)**

Invierte los bytes de la clave a indexar. Esto sirve para los índices cuyas claves son una serie constante con por ej. Crecimiento ascendente. para que las inserciones se distribuyan por todas las hojas del árbol de índice.

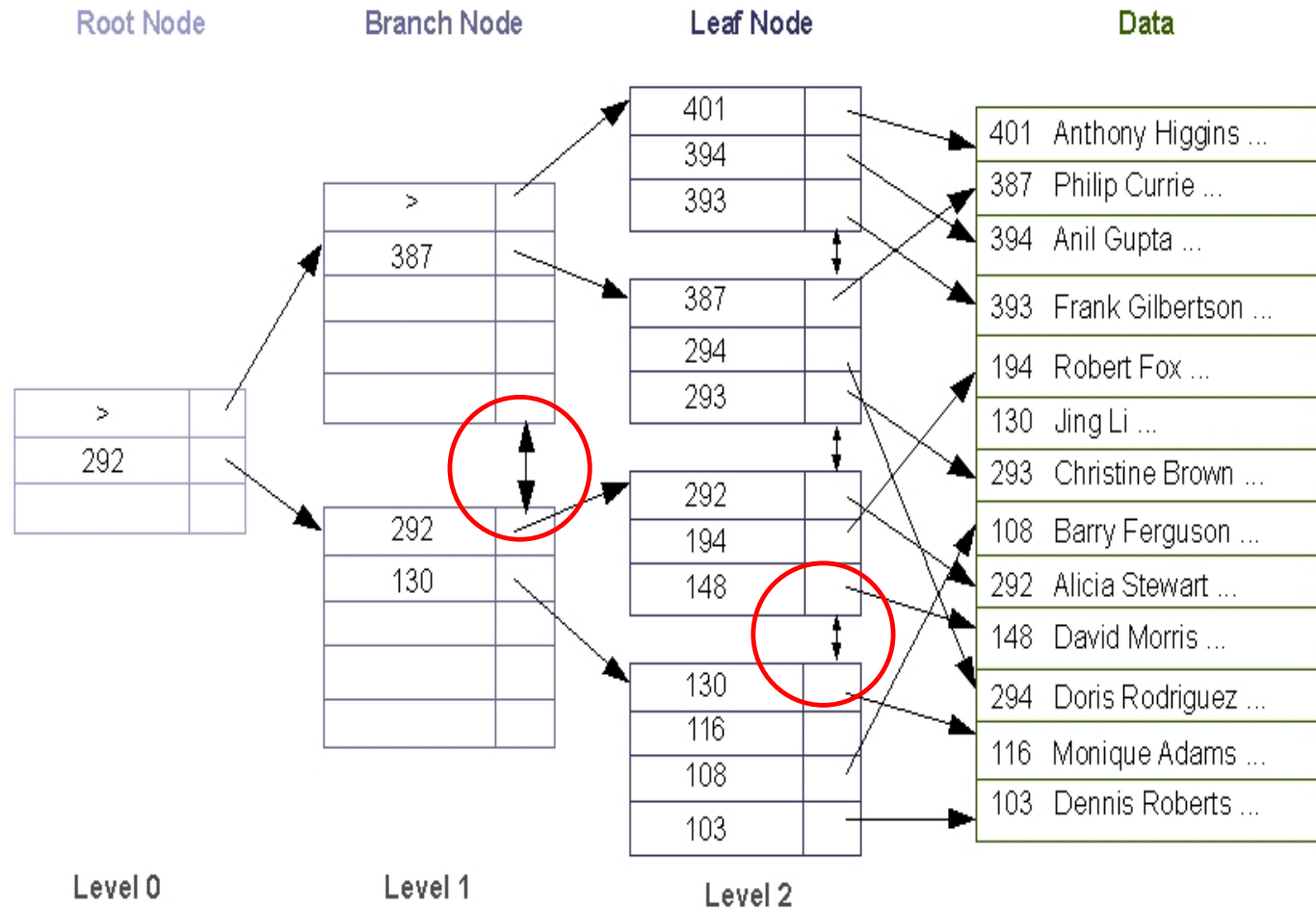
# Indices (Cont.)

## Indices B Tree



# Indices (Cont.)

## Indices B Tree +



# Indices (Cont.)

## CARACTERÍSTICAS DIFERENCIADORAS PARA LOS ÍNDICES

**Unique** Índice de clave única. Sólo admite una fila por clave.

**Duplicado** Permite múltiples filas para una misma clave.

**Simple** La clave está integrada por una sola columna.

**Compuesto** La clave se compone de varias columnas.  
Las principales funciones de un índice compuesto son:  
Facilitar múltiples joins entre columnas  
Incrementar la unicidad del valor de los índices

```
CREATE INDEX ix_sample  
ON sample_table (a, b, c);
```

a	b	c
a	b	c
a	b	c

a	b	c
a	b	c
a	b	c
a	b	c



# Indices (Cont.)

## BENEFICIOS DE LA UTILIZACIÓN DE INDICES:

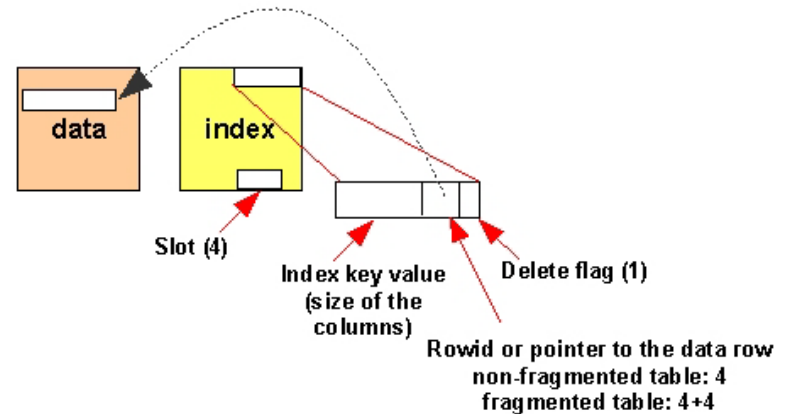
- Se le provee al sistema mejor performance al equipo ya que no debe hacer lecturas secuenciales sino accede a través de los índices, sólo en los casos que las columnas del Select no formen parte del índice.
- Mejor performance en el ordenamiento de filas
- Asegura únicos valores para las filas almacenadas
- Cuando las columnas que intervienen en un JOIN tienen índices se le da mejor performance si el sistema logra recuperar los datos a través de ellas
- Asegura el cumplimiento de constraints y reglas de negocio.
  - Primary key, foreign keys, unique values

## Indices (Cont.)

## COSTO DE LA UTILIZACIÓN DE INDICES

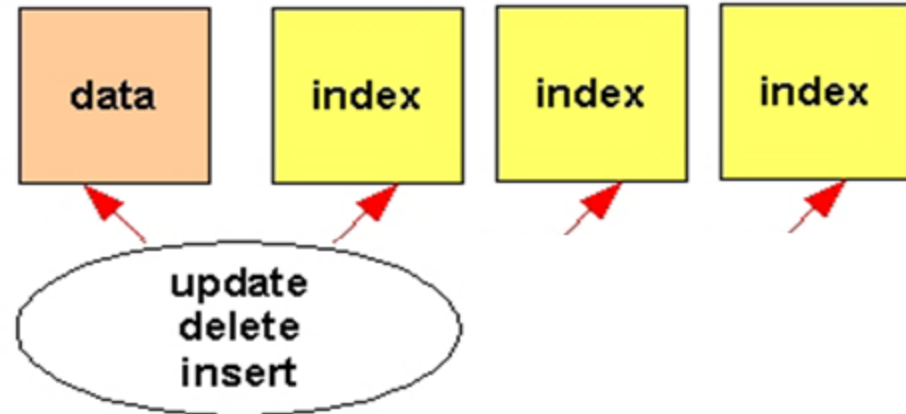
## COSTO DE ESPACIO DE DISCO

El primer costo asociado es el espacio que ocupa en disco, que en algunos casos suele ser mayor al que ocupan los datos.



## COSTO DE PROCESAMIENTO Y MANTENIMIENTO

El segundo costo es el de procesamiento, hay que tener en cuenta que cada vez que una fila es insertada o modificada o borrada, el índice debe estar bloqueado, con lo cual el sistema deberá recorrer y actualizar los distintos índices.



# Indices (Cont.)

## **GUIA RESUMEN CUANDO DEBERIAMOS INDEXAR**

- Indexar columnas que intervienen en Joins
- Indexar las columnas donde frecuentemente se realizan filtros
- Indexar columnas que son frecuentemente usadas en orders by
- Evitar duplicación de índices
  - Sobre todo en columnas con pocos valores diferentes Ej: Sexo, Estado Civil, Etc.
- Verificar que el tamaño de índice debería ser pequeño comparado con la fila
  - Tratar sobre todo en crear índices sobre columnas cuya longitud de atributo sea pequeña
  - No crear índices sobre tablas con poca cantidad de filas, no olvidar que siempre se recupera de a páginas. De esta manera evitaríamos que el sistema lea el árbol de índices

# Indices (Cont.)

## GUIA RESUMEN CUANDO DEBERIAMOS INDEXAR (Cont.)

- Limitar la cantidad de índices en tablas que son actualizadas frecuentemente
  - Porque sobre estas tablas se estarán ejecutando Selects extras
- Tratar de usar índices compuestos para incrementar los valores únicos
  - Tener en cuenta que si una o más columnas intervienen en un índice compuesto el optimizador podría decidir acceder a través de ese índice aunque sea solo para la búsqueda de los datos de una columna, esto se denomina "*partial key search*"
- Usando cluster index se agiliza la recuperación de filas
  - Uno de los principales objetivos de la Optimización de bases de datos es reducir la entrada/salida de disco. Reorganizando aquellas tablas que lo necesiten, se obtendría como resultado que las filas serían almacenadas en bloques contiguos, con lo cual facilitaría el acceso y reduciría la cantidad de accesos ya que recuperaría en menos páginas los mismos datos.

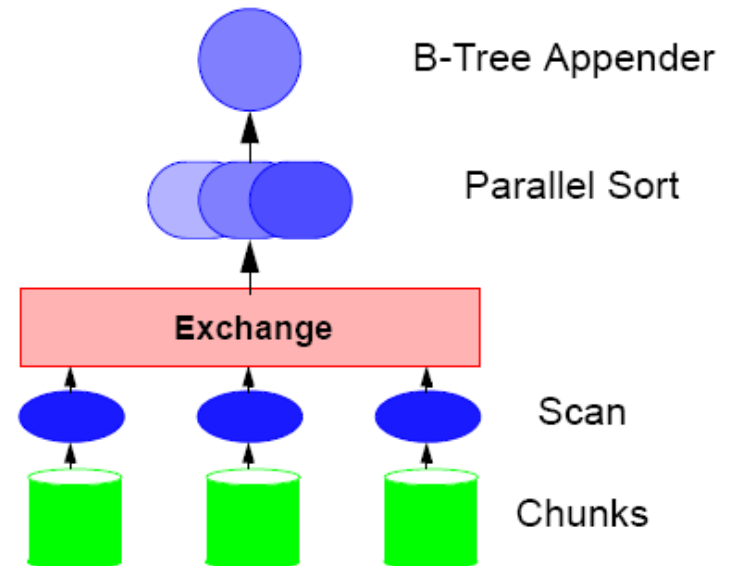
# Indices (Cont.)

## CONSTRUCCIÓN DE ÍNDICES EN PARALELO

Los motores de BD usan en general métodos de construcción de índices en paralelo.

El arbol B+ es construido por 2 o más procesos paralelos. Para esto el motor realiza una muestra de la filas a Indexar (aproximadamente 1000) y luego decide como separar en grupos. Luego scanea las filas y las ordena usando el mecanismo de sort en paralelo.

Las claves ordenadas son colocadas en los grupos apropiados para luego ir armarndo en paralelo un subárbol por cada grupo. Al finalizar los subárboles se unen en un único Arbol B+.



# Indices (Cont.)

## Motor Informix

Informix utiliza una estructura de Arbol B+

Máxima cantidad de campos para la clave de un índice compuesto son 16 y el tamaño de clave máximo permitido es 255 bytes.

Creación de Índice único y simple.

**CREATE UNIQUE INDEX ix1\_ordenes ON ordenes (n\_orden);**

Creación de Índice duplicado y compuesto.

**CREATE INDEX ix2\_ordenes ON ordenes (n\_cliente, f\_orden);**

Informix permite con el mismo índice realizar consultas ascendentes o descendentes indistintamente, sin requerir de ninguna cláusula adicional.

Creación de Índice cluster.

**CREATE CLUSTER INDEX ix3\_ordenes ON ordenes (f\_orden);**

El motor ordena los datos de la tabla igual que el índice. Sólo puede haber un índice cluster por tabla.

# Indices (Cont.)

## Motor Informix (Cont)

Creación de índice basado en función de Usuario:

```
CREATE FUNCTION myupper (v_value char(15))  
RETURNING char(15) with (not variant);  
define r_value char(15);  
execute function upper(v_value) into r_value;  
return r_value;  
END FUNCTION;
```

```
CREATE INDEX upper_ix1 ON state (myupper (sname));
```

Manejo del Load Factor

FILLFACTOR – Porcentaje de cada página del índice a ser llenado sólo en el momento de su creación.

Por ej. Si el FILLFACTOR=80%, en la creación del índice se ocupará hasta el 80% de cada nodo.

```
CREATE INDEX ix4_ordenes ON ordenes (n_cliente) FILLFACTOR 80;
```

# Indices (Cont.)

## Motor DB2

DB2 utiliza una estructura de Arbol B+

Máxima cantidad de campos para la clave de un índice compuesto son 16 y el tamaño de clave máximo permitido es 1024 bytes.

Creación de Índice único y simple, con campos de consulta adicionales incluidos.

```
CREATE UNIQUE INDEX ix1_ordenes  
ON ordenes (n_orden)  
INCLUDE (f_orden, c_cliente);
```

Los campos f\_orden y c\_cliente no conforman la clave del índice, pero se guardan dentro para acelerar las consultas, obteniendo dicha información del índice, sin necesidad de buscarla en la tabla.

Creación de Índice duplicado y compuesto.

```
CREATE INDEX ix2_ordenes  
ON ordenes (n_cliente, f_orden)  
ALLOW REVERSE SCANS;
```

DB2 requiere de la cláusula "ALLOW REVERSE SCANS" para poder usar el índice en consultas descendentes. Esta cláusula convierte al índice en un índice bi-direccional.

```
SELECT * FROM ordenes ORDER BY n_cliente DESC
```



# Indices (Cont.)

## Motor DB2 (Cont.)

### Creación de Indice cluster.

DB2 no posee índices cluster pero permite reorganizar físicamente una tabla en función a un determinado índice.

**CREATE INDEX ix3\_ordenes ON ordenes (f\_orden);**

**REORG TABLE ordenes INDEX ix3\_ordenes;**

### Manejo del Load Factor

PCTFREE – Porcentaje de cada página del índice a ser dejado como espacio libre en su creación.

Por ej. Si el PCTFREE=20%, en la creación del índice se ocupará hasta el 80% de cada nodo.

**CREATE INDEX ix4\_ordenes ON ordenes (n\_cliente) PCTFREE 20;**

# Indices (Cont.)

## Motor ORACLE

Oracle utiliza una estructura de Arbol B+.

Máxima cantidad de campos para la clave de un índice compuesto: 32.

Tamaño máximo de clave permitido: aproximadamente la mitad del espacio disponible en el bloque de datos.

Creación de un índice único y simple:

- **CREATE UNIQUE INDEX ix1\_ordenes ON ordenes (n\_orden);**

Creación de Indice duplicado y compuesto.

**CREATE INDEX ix2\_ordenes ON ordenes (n\_cliente, f\_orden);**

Creación de Indice cluster.

Oracle no posee índices cluster o clustered, pero posee el objeto IOT que tiene una estructura interna similar.

**Ver Objeto Tabla Organizada por índices (IOT - Index-Organized Tables)**

# Indices (Cont.)

## Motor ORACLE (cont.)

Creación de índice basado en función:

```
CREATE INDEX upper_ix ON ordenes (UPPER(d_usuario));
```

Creación de un índice bitmap:

```
CREATE BITMAP INDEX estado_bm_ix ON ordenes(c_estado)
```

Manejo del Load Factor

PCTFREE – Porcentaje de cada página del índice a ser dejado como espacio libre en su creación.

Por ej. Si el PCTFREE=20, en la creación del índice se ocupará hasta el 80% de cada nodo.

```
CREATE INDEX ix4_ordenes ON ordenes (n_cliente) PCTFREE 20;
```

# Indices (Cont.)

## Motor SQL SERVER

Sql Server utiliza una estructura de Arbol B+.

Máxima cantidad de campos para la clave de un índice compuesto: 16.

Creación de un índice único y simple:

```
CREATE UNIQUE INDEX ix1_ordenes ON ordenes (n_orden);
```

Creación de Indice duplicado y compuesto.

```
CREATE INDEX ix2_ordenes ON ordenes (n_cliente, f_orden);
```

Creación de Indice cluster.

```
CREATE CLUSTERED INDEX ix3_ordenes ON ordenes(N_orden) ON [PRIMARY];
```

# Indices (Cont.)

## Motor SQL SERVER (cont.)

Manejo del Load Factor

FILLFACTOR– Porcentaje de cada página del índice a ser dejado como espacio libre en su creación.

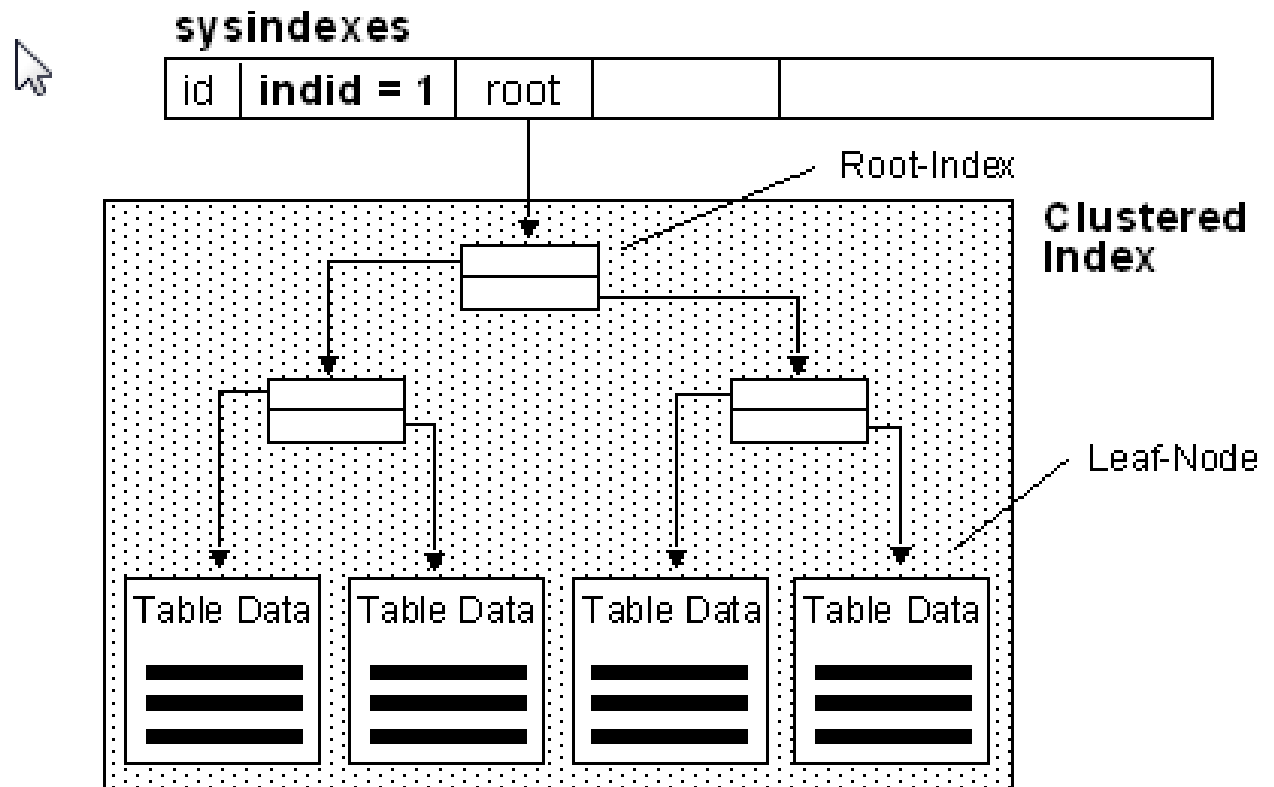
Por ej. Si el FILLFACTOR=20, en la creación del índice se ocupará hasta el 80% de cada nodo.

```
CREATE UNIQUE INDEX ix1_ordenes ON ordenes(N_orden)  
WITH FILLFACTOR = 20;
```

# Indices (Cont.)

## MOTOR SQL SERVER

### Indices Cluster



# Tablas Organizadas por Índice (IOT - Index-Organized Tables) (Ora)

Una IOT tiene una organización del almacenamiento que es una variante del Árbol-B. A diferencia de una tabla común, en la que los datos se guardan como un conjunto desordenado, en una IOT se guardan en una estructura de índice de Árbol-B, ordenado a la manera de una clave primaria. Sin embargo, además de almacenar los valores de las columnas de clave primaria de cada tabla, cada entrada en el índice almacena además los valores de las columnas no clave.

De esta forma, en vez de mantener dos estructuras separadas de almacenamiento, una para la tabla y una para el índice de Árbol-B, el sistema mantiene sólo un índice Árbol-B, porque además de almacenar el rowid de las filas, también se guardan las columnas no clave.

# Tablas Organizadas por Índice (IOT - Index-Organized Tables) (Ora)

## Motor Oracle

La siguiente sentencia crea la tabla ordenes, como una tabla organizada por índice, indicando que la columna N\_cliente divide el área del índice, de las columnas no clave.

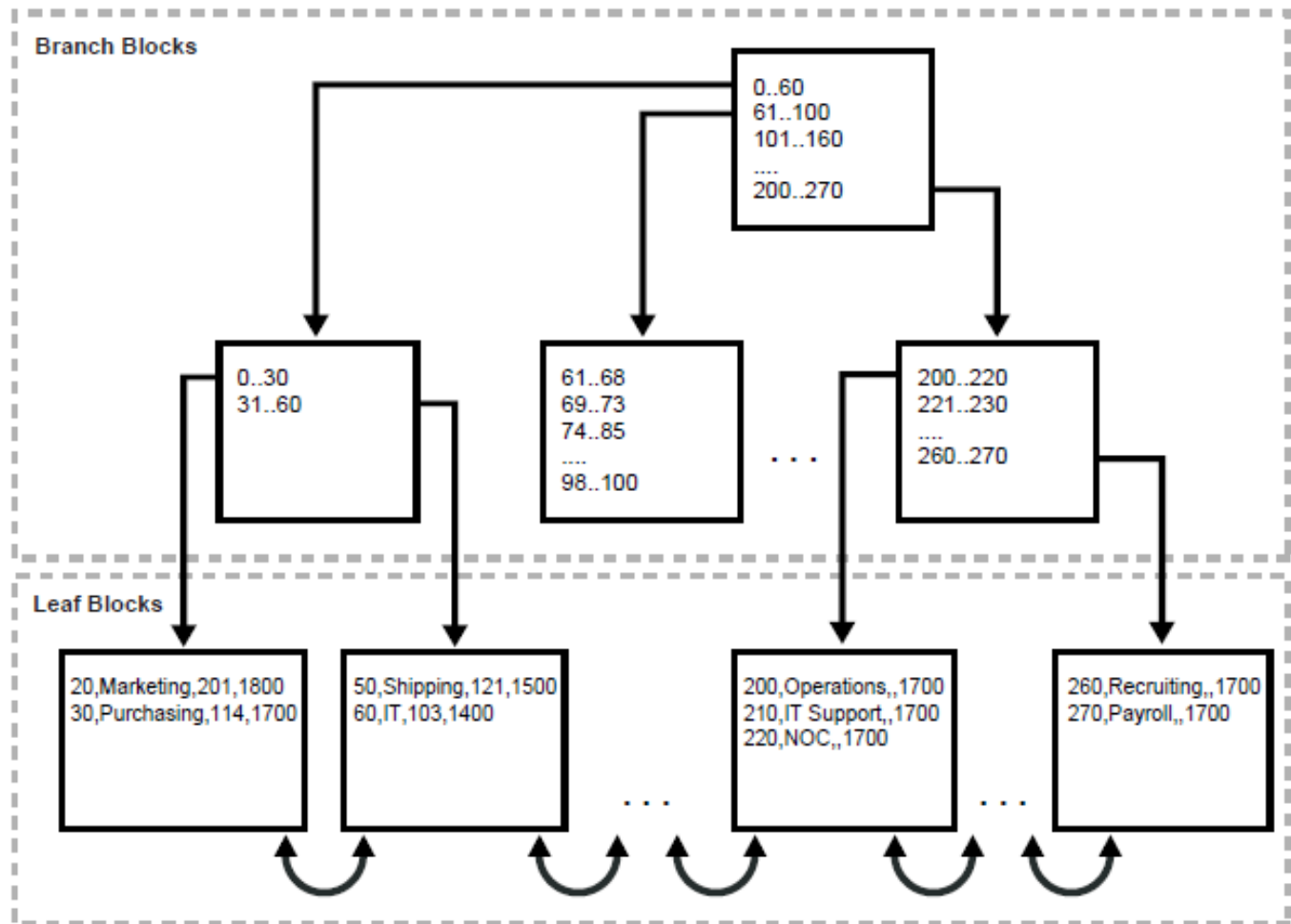
```
CREATE TABLE ordenes  
( N_orden NUMBER PRIMARY KEY,  
  N_cliente NUMBER,  
  F_orden DATE,  
  C_estado NUMBER,  
  F_alta_audit DATE,  
  D_usuario VARCHAR2(20) )
```

**ORGANIZATION INDEX INCLUDING N\_CLIENTE OVERFLOW**



# Tablas Organizadas por Índice (IOT - Index-Organized Tables) (Ora)

Motor Oracle



# Sinónimos / Nicknames

Un sinónimo es un alias definido sobre una tabla, vista ó snapshot. Dependiendo el motor de BD puede ser también definido sobre un procedure, una secuencia, función o package.

Los sinónimos se usan a menudo por seguridad o por conveniencia (por ej. en entornos distribuidos).

Permiten enmascarar el nombre y dueño de un determinado objeto.  
Proveen de una ubicación transparente para objetos remotos en una BD distribuida.

Simplifican las sentencias SQL para usuarios de la BD.

# Sinónimos / Nicknames (Cont.)

## Motor Informix

Creación de un sinónimo con el nombre ordenes\_cordoba en la BD\_comercial en Buenos Aires que apunta a la tabla ordenes de la BD bd\_comercial del Servidor en Córdoba.

```
DATABASE bd_comercial@server_bsas;  
CREATE SYNONYM ordenes_cordoba FOR  
bd_comercial@server_cordoba:ordenes
```

## Motor Oracle

Creación de un sinónimo público con el nombre ordenes\_cordoba en la bd\_bsas, que apunta a la tabla ordenes de la bd\_cordoba, propiedad del usuario admin.

Previamente se deberá crear un enlace de Base de Datos (DBLink) que permita ver la bd\_cordoba desde la bd\_bsas.

```
CREATE PUBLIC SYNONYM ordenes_cordoba FOR  
admin.ordenes@bd_cordoba;
```

# Sinónimos / Nicknames (Cont.)

## Motor DB2

Creación de un nickname en Bases de Datos Federadas. Un Sistema Federado de base de datos es un tipo de especial de sistemas de base de datos distribuidas.

Se crea un nickname con el nombre ordenes\_cordoba en la BD\_comercial en Buenos Aires que apunta a la tabla ordenes de la BD bd\_comercial del Servidor en Córdoba.

Previamente se deberá crear una definición de un server en la que se debe especificar el usuario y password que puede conectarse a la fuente remota y el nombre del server al que se hará referencia.

El nickname puede ser creado para acceder localmente a una tabla que se encuentra en un sitio remoto.

```
CREATE SERVER server_cordoba  
TYPE DB2/UDB VERSION '8.1'  
WRAPPER "DRDA"  
AUTHID "PEPE"  
PASSWORD "*****"  
OPTIONS ( DBNAME 'bd_comercial' );
```

```
CREATE USER MAPPING FOR USER  
SERVER server_cordoba  
OPTIONS ( REMOTE_AUTHID 'PEPE'  
REMOTE_PASSWORD '*****');
```

```
CREATE NICKNAME ordenes_cordoba FOR .server_cordoba.pepe.ordenes;
```

# Database Links (Oracle)

Un database link es un puntero que define una ruta de comunicación unidireccional desde un servidor de base de datos hasta otro.

Para acceder al enlace, se debe estar conectado a la base de datos local que contiene la entrada en el diccionario de datos que define dicho puntero.

Hay dos tipos de enlace según la forma en que ocurre la conexión a la base remota:

- Enlace de usuario conectado: el usuario debe tener una cuenta en la base remota con el mismo nombre de usuario de la base local.
- Enlace de usuario fijo: el usuario se conecta usando el nombre de usuario y password referenciados en el enlace.

# Database Links (Oracle) (Cont.)

## Motor Oracle

Creación de un database link público, llamado "remota", que hace referencia a la base de datos especificada por el nombre de servicio "ventas":

```
CREATE PUBLIC DATABASE LINK remota  
USING 'ventas';
```

Uso del database link en una actualización de la tabla remota ordenes para el cliente 200.

```
UPDATE ordenes@remota  
SET c_estado = 0  
WHERE n_cliente = 200;
```

Creación de un database link privado con usuario fijo admin/admin, a la base de datos especificada por el nombre de servicio "ventas".

```
CREATE DATABASE LINK remota  
CONNECT TO admin IDENTIFIED BY admin  
USING 'ventas';
```

# Directories (Oracle)

Un directorio especifica un alias para un directorio en el file system del servidor, donde se ubican archivos binario externos (BFILES) y datos de tablas externas.

Se pueden usar nombres de directorios al referirse a ellos desde código PL/SQL, en vez de hardcodear el nombre de ruta de sistema operativo, suministrando por lo tanto una mayor flexibilidad en la administración de archivos. Los directorios no son propiedad de ningún esquema individual.

## Motor Oracle

```
CREATE DIRECTORY fuentes AS '/sistema/usr/fuentes';
```

# Clusters (Oracle)

Un cluster es un **grupo de tablas** que **comparten los mismos bloques de datos** porque tienen **columnas comunes compartidas** y que a menudo se usan juntas. Cuando se agrupan tablas, el motor guarda físicamente todas las filas de cada una de ambas tablas en los mismos bloques de datos.

Beneficios:

- Reducción de E/S a disco para los joins de las tablas agrupadas
- Mejora en tiempos de acceso para los joins de las tablas agrupadas
- Requiere menos espacio para almacenar tablas relacionadas e índices de datos.



# Clusters (Oracle)

## Creación de cluster con clave N ORDEN.

```
CREATE CLUSTER ventas  
(n_orden NUMBER)  
SIZE 512  
STORAGE (initial 100K next 50K);
```

## Creación de índice de cluster.

```
CREATE INDEX idx_ventas ON CLUSTER  
ventas;
```

## Creación de tablas agregadas al cluster

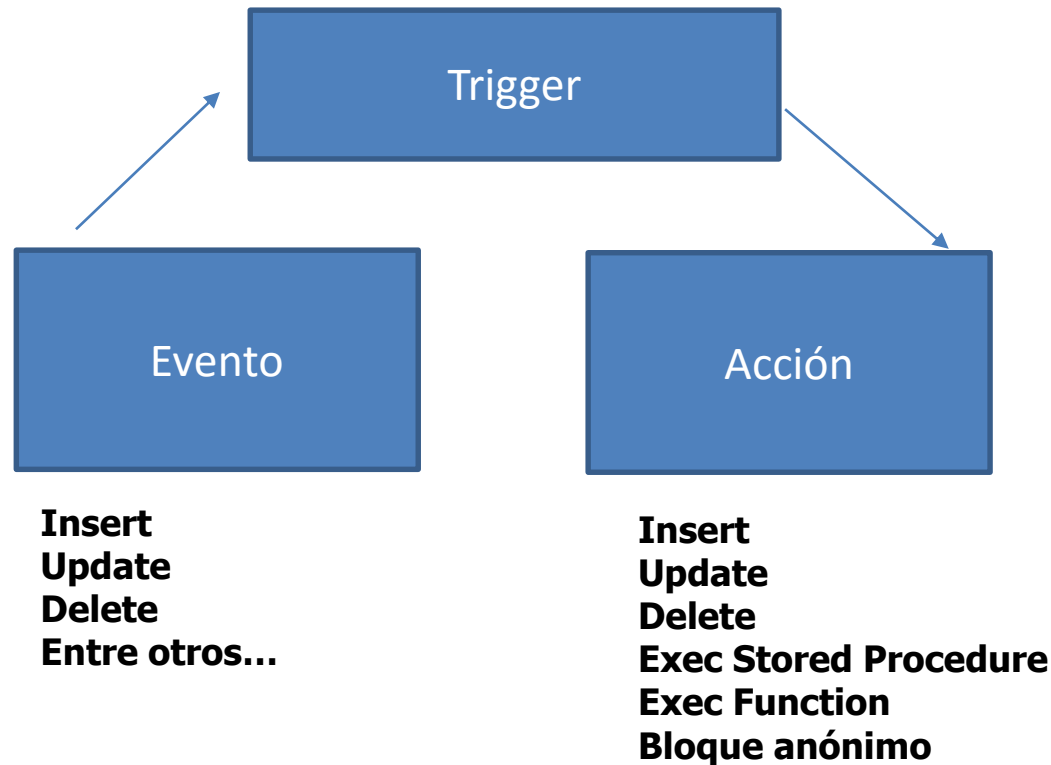
```
CREATE TABLE ordenes  
( N_orden NUMBER PRIMARY KEY,  
  N_cliente NUMBER,  
  F_orden DATE,  
  C_estado NUMBER,  
  F_alta_audit DATE,  
  D_usuario VARCHAR2(20) )  
CLUSTER ventas (n_orden);
```

```
CREATE TABLE item_ordenes  
( N_orden NUMBER REFERENCES ordenes,  
  N_item NUMBER,  
  C_producto NUMBER,  
  Q_cantidad NUMBER,  
  I_precunit NUMBER(9,3),  
  PRIMARY KEY (n_orden, n_item) )  
CLUSTER ventas (n_orden);
```

# Triggers

Qué es un Trigger?

Es un mecanismo que ejecuta una sentencia de SQL automáticamente cuando cierto evento ocurre.



# Triggers

## PORQUE USAR TRIGGERS?

Se pueden aplicar las reglas de negocio

- Por Ej.: Si el inventario de una columna pasa x valor, entonces insertar un pedido en la tabla de compras
- Valores de columnas derivadas
- En algunos casos es necesario que ante tal acción se deriven datos en base a otros.
- Replicación automática de tablas
- Logs. De Auditoría
- Delete en Cascada
- Autorización de Seguridad

# Triggers

## Eventos Posibles

### Instrucciones DML sobre Tablas o Views

- INSERT ON tab\_name
- DELETE FROM tab\_name
- UPDATE tab\_name
- UPDATE of col\_name ON tab\_name
- SELECT tab\_name (Informix)
- SELECT of col\_name ON tab\_name (Informix)

### Instrucciones DDL sobre Base de Datos (Oracle/Sql Server)

- CREATE
- ALTER
- DROP

# Triggers

## Eventos Posibles (Cont.)

### Operaciones de Base de Datos (Oracle/Sql Server)

- SERVERERROR
- LOGON
- LOGOFF
- STARTUP
- SHUTDOWN

En algunos motores se permiten múltiples triggers sobre una tabla, pero sólo 1 por tipo. Para el caso de UPDATE las columnas deben ser mutuamente exclusivas.

La tabla especificada por el trigger debe estar en modo local, en general los motores no aceptan tablas en servers remotos.

# Funciones Propias del Motor (Built-in Functions)

Estos objetos son funciones ya desarrolladas con el Motor de BD, las cuales pueden clasificarse de la siguiente manera:

## Funciones Agregadas

Se aplican a un conjunto de valores derivados de una expresión, actúan específicamente en agrupamientos.

SUM, COUNT, AVG, MAX, MIN, etc.

## Funciones Escalares

Se aplican a un dato específico de cada fila de una consulta, o en una comparación en la sección WHERE.

Funciones Algebraicas/Trigonométricas

Funciones Estadística

Funciones de Fecha

Funciones de Strings

Funciones de Conversión

Funciones de Manejo de Null

Funciones de Entorno

Funciones XML (Ora)

Funciones de Minería de Datos (Ora)  
entre otras.

## Funciones de Tablas (algunos motores)

Retornan el equivalente a una tabla y pueden ser usadas solamente en la sección FROM de una sentencia.

ORA: Se denominan "Vista en línea (inline view)"

# Procedimientos Almacenados

Es un procedimiento programado en un lenguaje permitido que es almacenado en la Base de Datos como un objeto. El mismo luego de creado, puede ser ejecutado por usuarios que posean los permisos respectivos.

## Características Principales

- Incluyen sentencias de SQL y sentencias de lenguaje propias. Lenguaje SPL (Informix), PL/SQL (Oracle), TRANSAC/SQL (SQL Server).
- Son **almacenados en la base de Datos**
- **Algunos** motores permiten además **Stored Procedures en JAVA**.
- Antes de ser almacenada en la base de datos las sentencias **SQL son parseadas y optimizadas**. Cuando el “stored procedure” es ejecutado puede que no sea necesario su optimización, en caso contrario se optimiza la sentencia antes de ejecutarse

# Procedimientos Almacenados (Cont.)

## VENTAJAS DE LOS STORED PROCEDURES

- Pueden **reducir la complejidad en la programación**. Creando SP con las funciones más usadas.
- Pueden **ganar performance en algunos casos**.
- Otorgan un **nivel de seguridad extra**.
- Pueden definirse **ciertas reglas de negocio independientemente de las aplicaciones**.
- **Diferentes aplicaciones** acceden al **mismo código ya compilado y optimizado**.
- En una **arquitectura cliente/servidor**, no sería necesario tener **distribuido el código** de la aplicación
- En proyectos donde el **código puede ser ejecutado desde diferentes interfaces**, Ud. mantiene un solo tipo de código.
- **Menor tráfico en el PIPE / SOCKET**, no en la cantidad de bytes que viajan sino en los ciclos que debo ejecutar una instrucción.



# Procedimientos Almacenados (Cont.)

Creación de un Stored Procedure que inserta una orden de pedido a partir de los datos existentes en varias tablas temporales manejando una transacción y creación de un procedure de grabación de log de errores.

## Ejemplo en Transac/Sql - Motor SqlServer

```
CREATE PROCEDURE dbo.sp_inserta_orden
@v_n_orden INT
AS
SET NOCOUNT ON
DECLARE
    @nError int,
    @error_info char(30)
SET @nError = 0

BEGIN TRANSACTION

INSERT INTO ordenes
SELECT * FROM ordenes_tmp
WHERE n_orden = @v_n_orden

INSERT INTO item_ordenes
SELECT * FROM items_tmp
WHERE n_orden = @v_n_orden
set @nError = @@error

IF( @nError <> 0 )
BEGIN
    ROLLBACK
    SELECT @nError AS nError,
           @error_info AS Descripcion

    exec sp_error_log @nError,
                      @error_info,
                      @v_n_orden,
                      'sp_inserta_orden'
END
ELSE
BEGIN
    COMMIT
    SELECT 0 AS nError ,
           'Se insertó bien' AS Descripcion
END
GO
```

# Procedimientos Almacenados (Cont.)

Creación de un Stored Procedure que inserta una orden de pedido a partir de los datos existentes en varias tablas temporales manejando una transacción y creación de un procedure de grabación de log de errores.

## Ejemplo en Transac/Sql - Motor SqlServer (Cont.)

```
CREATE PROCEDURE  dbo.sp_error_log
@sql_err int,
@error_info char(70),
@v_nro_orden INT,
@proc_name char(18)

AS

SET NOCOUNT ON

INSERT INTO error_logs
VALUES (@proc_name, @v_nro_orden,
@sql_err,@error_info, USER, getdate())

GO
```

# Procedimientos Almacenados (Cont.)

Creación de un Stored Procedure que inserta una orden de pedido a partir de los datos existentes en varias tablas temporales manejando una transacción y creación de un procedure de grabación de log de errores.

## Ejemplo en SPL - Motor Informix

```
CREATE PROCEDURE sp_inserta_orden (V_n_orden INT)
RETURNING SMALLINT;
```

```
    DEFINE sql_err int;
    DEFINE isam_err int;
    DEFINE error_info char(70);
```

```
    ON EXCEPTION SET sql_err, isam_err, error_info
        ROLLBACK WORK;
        CALL error_log (sql_err, isam_err, error_info,
                        V_n_orden, "sp_inserta_orden");
```

```
        RAISE EXCEPTION sql_err, isam_err, error_info;
    END EXCEPTION;
```

```
BEGIN WORK;
INSERT INTO ordenes
SELECT * FROM ordenes_tmp
WHERE n_orden = V_n_orden;
```

```
INSERT INTO items_ordenes
SELECT * FROM items_tmp
WHERE n_orden = V_n_orden;
```

```
COMMIT WORK;
RETURN 0;
END PROCEDURE;
```

# Procedimientos Almacenados (Cont.)

Creación de un Stored Procedure que inserta una orden de pedido a partir de los datos existentes en varias tablas temporales manejando una transacción y creación de un procedure de grabación de log de errores.

## Ejemplo en SPL - Motor Informix (Cont.)

```
CREATE PROCEDURE sp_error_log (sql_err int, isam_err int,  
                                error_info char(70),  
                                nro_orden INT,  
                                proc_name char(18) )  
  
    INSERT INTO error_logs  
    VALUES (proc_name, v_nro_orden, sql_err, isam_err,  
            error_info, USER, CURRENT);  
  
END PROCEDURE;
```

# Funciones de Usuario

Una función de usuario es un objeto de la base de datos programado en un lenguaje válido por el motor de base de datos que puede recibir uno o más parámetros de input y devolver sólo un parámetro de output.

## Ejemplo Motor Oracle

```
CREATE FUNCTION nombre_dpto (p_c_empleado NUMBER)
RETURN departamentos.d_depto%TYPE;
DECLARE
v_nombre_dpto      departamentos.d_depto%TYPE;
v_error            VARCHAR2(70);
BEGIN
    SELECT d_dpto INTO v_nombre_dpto
    FROM departamentos d, empleados e
    WHERE d.c_dpto = e.c_dpto
    AND e.c_empleado = p_c_empleado;

    RETURN v_nombre_dpto;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    v_error := 'Error: empleado ' || p_c_empleado || ' no se ha encontrado';
    raise_application_error(-20101, v_error);

END nombre_dpto;
```

# Funciones de Usuario (Cont.)

Una función de usuario es un objeto de la base de datos programado en un lenguaje válido por el motor de base de datos que puede recibir uno o más parámetros de input y devolver sólo un parámetro de output.

## Ejemplo Motor Oracle (Cont.)

Invocación de función en un Select en la sección WHERE

```
SELECT *  
FROM departamentos d  
WHERE d.d_dpto = nombre_dpto(6010);
```

Invocación de función en un Select en la sección Lista de Columnas

```
SELECT e.c_empleado, e.d_nombre, e.d_apellido, nombre_dpto(e.c_dpto) depto_nombre  
FROM empleados e  
WHERE e.c_empleado = 6010;
```

# Funciones de Usuario (Cont.)

Una función de usuario es un objeto de la base de datos programado en un lenguaje válido por el motor de base de datos que puede recibir uno o más parámetros de input y devolver sólo un parámetro de output.

## Ejemplo Motor Oracle (Cont.)

Invocación de función en un Select en la sección WHERE

```
SELECT *  
FROM departamentos d  
WHERE d.d_dpto = nombre_dpto(6010);
```

Invocación de función en un Select en la sección Lista de Columnas

```
SELECT e.c_empleado, e.d_nombre, e.d_apellido, nombre_dpto(e.c_dpto) depto_nombre  
FROM empleados e  
WHERE e.c_empleado = 6010;
```

# Packages (Oracle / DB2)

Un Package (paquete) es un objeto que agrupa tipos y subprogramas (funciones y procedimientos almacenados) relacionados lógicamente. Habitualmente tienen dos partes: una especificación y un cuerpo.

La especificación es la interfaz para las aplicaciones: declara los tipos, variables, constantes, excepciones, cursores y subprogramas disponibles para su uso.

El cuerpo define en forma completa los cursores y subprogramas, y por lo tanto implementa la especificación.

Los paquetes suministran varias ventajas: modularidad, facilidad en el diseño de la aplicación, ocultamiento de información y mejora en el rendimiento.



# Esquemas (Oracle)

Oracle define esquema como la colección de objetos o estructuras lógicas que corresponden directamente a los datos almacenados, y **crea un nuevo esquema por cada usuario** que crea objetos en la base de datos.

Oracle crea automáticamente un esquema cuando se crea un Nuevo usuario.

Los esquemas están agrupados por usuario de base de datos e incluyen tablas, vistas, y otros objetos.