Clase 1:
Concepto de base de datos

Colección o conjuntos de datos interrelacionados con un propósito específico vinculado a un problema del mundo real. Un dato representa hechos conocidos que pueden registrarse y tienen un resultado implícito.

Propiedades implícitas de una base de datos

- Una BD representa algunos aspectos del mundo real (a veces denominado *universo de discurso*)
- Una BD es una colección coherente de datos con significado. Un conjunto aleatorio de datos no puede ser considerado una BD. Los datos deben tener cierta lógica.
- Una BD se diseña, construye y completa con datos para un propósito específico. Está destinada a un grupo de usuarios concretos.
- Una BD está sustentada fisicamente en archivos sobre dispositivos de almacenamiento persistente de datos.

Conceptos básicos

La **definición** de una BD consiste en especificar los tipos de datos, las estructuras y las restricciones de los mismos.
La **construcción** de la BD es el proceso de almacenar datos en algún dispositivo de almacenamiento bajo la gestión de un *DBMs (*)*.
La **manipulación** de la BD incluye funciones tales como: recuperación de datos, actualizar datos existentes, agregar nuevos datos, etc.

(*) Un **DBMS** es una colección de programas que permiten a los usuarios crear y mantener la base de datos. Es un software de propósito general, que facilita los procesos de construcción y manipulación.

Objetivos de un DBMS

- Evitar la redundancia e inconsistencia de datos.
- Permitir el acceso a los datos en todo momento.
- Evitar anomalías en el acceso concurrente.
- Seguridad: restricción a accesos no autorizados.
- Integridad de datos.
- Backups.

Componentes de un DBMS

- **DDL** (data definition language): especifica el esquema de la BD. Por ejemplo: creación de tablas, eliminación de tablas, etc.
- **DML** (data manipulation language): Agregar información, quitar información, modificar información, etc.
  Se diferencian dos tipos:
  - *Procedimentales* (SQL): requieren que el usuario especifique qué datos se muestran y **cómo se deben obtener**.

- *No procedimentales* (QBE): requieren que el usuario especifique qué datos se muestran **sin especificar cómo deben ser obtenidos**.

Actores involucrados con una base de datos

- **DBA**: Administra la BD. Autoriza accesos, coordina y vigila la utilización de los recursos de hardware y software, responsable ante problemas de seguridad y respuesta lenta de la BD.
- **Diseñador de BD**: Definen la estructura de la BD de acuerdo al problema del mundo real que se esté solucionando.
- **Analistas de sistemas**: Determinan los requerimientos de los usuarios finales, generando la información necesaria para el diseñador.
- **Programadores**: Implementan las especificaciones de los analistas utilizando la BD generada por el diseñador.
- **Usuarios finales**

Conceptos más relevantes

- Aprender a definir una BD: Construcción del modelo de datos (diseño).
- Aprender a manipular una BD: Lenguaje de trabajo clásico con base de datos, como SQL.
- Estudio de seguridad e integridad de la información.

Modelado

Un modelo de datos es una colección de herramientas conceptuales para describir datos, relaciones entre ellos, semántica asociada a los datos y restricciones de consistencia. Los modelos de datos sirven para hacer más fácil la comprensión de los datos de una organización. Son medios que se utilizan para describir la realidad. Se modela para:

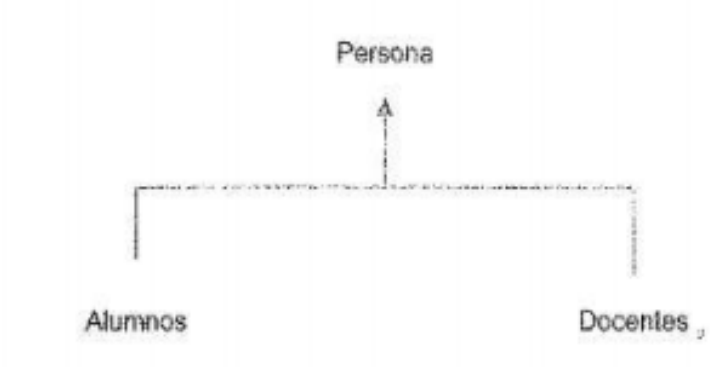
- Obtener la perspectiva de cada actor asociado al problema.
- Obtener la naturaleza y necesidad de cada dato.
- Observar cómo cada actor utiliza cada dato

Para construir un modelo se utilizan tres clases de abstracciones. La abstracción es un proceso que permite seleccionar algunas características importantes de un conjunto de objetos del mundo real, dejando de lado aquellos atributos que no son de interés para el problema.

- **Abstracción de clasificación:** Define una clase. Se origina a partir de las características comunes que tienen los objetos que la componen.
- **Abstracción de agregación:** Define una nueva clase a partir de otras clases que representan sus partes componentes. Por ejemplo, una clase *PC* está compuesta por otras clases como *CPU, Memoria RAM, HDD,* etc.
  Cuando se trabaja con agregaciones, es necesario identificar las correspondencias entre las distintas clases. Este concepto puede definirse como **cardinalidad.**
  Es posible definir el nivel mínimo de correspondencia (**cardinalidad mínima**) y el nivel máximo de correspondencia (**cardinalidad máxima**) entre las clases.
    - Cardinalidad mínima
        - Si es 0, indica que en la agregación hay una participación opcional.
        - Si es 1, la participación es obligatoria.

- Cardinalidad máxima
    - Si la card. max(a, b) = 1 y card. max(c, b) = 1, se dice que la cardinalidad es **uno a uno**.
    - Si la card. max(a, b) = 1 y card. max(c, b) = n, se dice que la cardinalidad es **uno a muchos.**
    - Si la card. max(a, b) = n y card. max(c, b) = 1, se dice que la cardinalidad es **muchos a uno.**
    - Si la card. max(a, b) = n y card. max(c, b) = n, se dice que la cardinalidad es **muchos a muchos.**

- **Abstracción de generalización:** Define una relación de subconjunto entre los elementos de dos o más clases.
  Cuando se trabaja con generalizaciones, es posible definir las **coberturas** de la misma:
    - Relación entre padre/hijo total o parcial:
        - Total: Cada elemento del padre está contenido en alguno de los hijos.
        - Parcial: Pueden existir elementos del padre que no estén en los hijos.
    - Cobertura exclusiva o superpuesta (evalúa si un elemento del padre puede o no estar en más de un hijo):
        - Exclusiva: solo puede estar en un hijo.
        - Superpuesta: puede estar en varios hijos.



  Tomando de ejemplo esta generalización:
    - Si un docente no puede ser alumno, y un alumno no puede ser docente, entonces la cobertura es **exclusiva**. Por otro lado, si pueden haber docentes que son alumnos, y alumnos docentes, la cobertura es **superpuesta.**
    - Si puede existir un elemento exclusivamente de la clase *Persona* la cobertura es **parcial.** Caso contrario la cobertura es **total.**

Tipos de abstracciones:
- **Visión**: Ve solo los datos de interés a través del programa de aplicación
- **Conceptual**: Qué datos se almacenan en la BD y qué relaciones existen entre ellos.
- **Físico**: Describe cómo se almacenan realmente los datos.

Técnicas de modelado y tipos de modelos
- **Basado en objetos** (visión **conceptual**): Estructura flexible, especifican restricciones explícitamente. Ejemplos:
    - *Modelo entidad-relación*: Presenta las entidades de datos, sus atributos asociados, y las relaciones entre estas entidades. Este modelo se basa en la

concepción del mundo real como un conjunto de objetos llamados entidades y las relaciones existentes entre dichas entidades.
    - Modelo orientado a objetos.
    - Etc.
- **Basado en registros** (conceptual físico): La BD se estructura en registros de formato fijo. Se dispone de un lenguaje asociado para expresar consultas. Ejemplos:
    - *Relacional:* El más representativo de los modelos basados en registros **(*)**
    - OO.
    - Jerárquico.
    - Red.
- **Físico de datos.**

NOTA (*): No confundir modelo relacional con modelo entidad-relación: **El modelo relacional utiliza un conjunto de tablas para representar las entidades y relaciones existentes, definidas previamente en el modelo entidad-relación.**

La estructura básica del modelo relacional es la **tabla**. Cada tabla tiene una estructura conformada por **columnas o atributos** que representan los mismos atributos definidos en el modelo entidad-relación.

Independencia de datos
Capacidad de modificar esquemas sin alterar otro nivel
- **Físico**: Modificar el esquema físico sin provocar que los programadores tengan que reescribir los programas de aplicación.
- **Lógico**: Modificar el esquema conceptual.

Categoría de software de procesamiento de datos
- Sin independencia de datos (Sistema operativo, transferencia a un sector en particular).
- Independencia física (leer un registro de un archivo)
- Independencia lógica parcial (leer siguiente registro de un archivo)
- Independencia lógica y física (leer siguiente registro de un tipo particular, DBMS)
- Independencia geográfica (base de datos distribuidas)

Diseño de datos
Consta de tres etapas:
1) Conceptual: representación abstracta.
2) Lógico: representación en una computadora.
3) Físico: determinar estructuras de almacenamiento físico.

Modelo Conceptual Entidad-Relación

Permite modelar el nivel conceptual y lógico de una BD, utilizando entidades e interrelacionandolas.

Objetivos:
- Representa la información de un problema en un alto nivel de abstracción.

- Captar la necesidad de un cliente respecto del problema que enfrenta.
- Mejora la interacción cliente/desarrollador disminuyendo la brecha entre la realidad del sistema y el sistema a desarrollar.

Características:
- **Expresividad**: disponer de todos los medios necesarios para describir un problema.
- **Formalidad**: que cada elemento representado sea preciso y bien definido, con una sola interpretación posible.
- **Minimalidad**: cada elemento tiene una representación posible.
- **Simplicidad**: el modelo debe ser fácil de entender por el cliente y por el desarrollador.

Componentes
- **Entidades**: representa un elemento u objeto del mundo real con **identidad.**
- **Conjunto de entidades:** es una representación que, a partir de las características propias de cada entidad con propiedades comunes, se resume en un núcleo.
- **Relaciones:** representan agregaciones entre dos o más entidades.
- **Conjunto de relaciones:** es una representación que, a partir de las características propias de cada relación existente entre dos entidades, las resume en un núcleo.
- **Atributos:** Representan una propiedad básica de una entidad o relación. Equivalen a un campo de un registro. Los atributos también tienen asociado el concepto de cardinalidad

  Cardinalidad:
  - Monovalente
  - Polivalente: Podrían existir múltiples valores para este atributo.
  - Obligatorio/opcional.

  Por defecto los atributos que esten definidos únicamente por su nombre tienen asociada la cardinalidad 1..1 (monovalente obligatorio)

Componentes adicionales
- **Identificadores**: Es un atributo o conjunto de atributos que permiten reconocer una entidad de manera unívoca. Pueden ser:
  - Simples o compuestos: de acuerdo a la cantidad de atributos que conforman al identificador.
  - Internos o externos: si todos los atributos pertenecen a la entidad que identifican, el identificador es interno. Caso contrario, es externo.
- **Atributos compuestos:** representan un atributo generado a partir de una combinación de atributos simples. Un atributo compuesto **puede ser polivalente y no obligatorio**. Sus **atributos simples pueden ser polivalentes y no obligatorios también.**
- **Jerarquías de generalización:** Permite extraer propiedades comunes de varias entidades o relaciones y generar con ellas una superentidad que las contenga. El concepto es similar al de abstracción por generalización.

Ventajas y desventajas del modelo conceptual
Las ventajas fueron nombradas en la sección de características. Por el lado de las desventajas:

- Lograr que un modelo sea expresivo puede atentar contra la simplicidad.
- La forma de expresar y determinar las cardinalidades e identificadores puede ser compleja.
- Las relaciones n-arias también presentan inconvenientes.

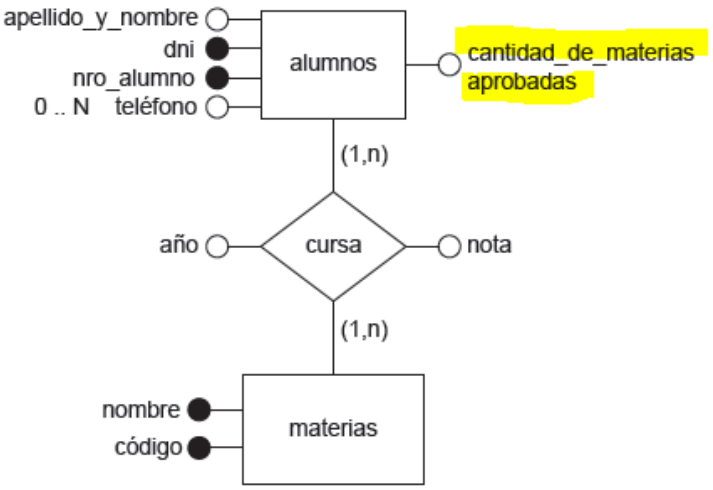
Clase 2:
Revisiones del modelo conceptual

Una vez finalizada la construcción del modelo conceptual, este debe ser puesto en consideración y análisis por parte del usuario. Es necesario lograr un modelo que represente la realidad del problema lo más estrictamente posible, disminuyendo las ambigüedades y aumentando la expresividad y legibilidad. Existen una serie de conceptos a revisar:

- **Expresividad**: El modelo resulta expresivo si a partir de su observación es posible darse cuenta de todos los detalles que lo involucran.
- **Autoexplicativo**: Un modelo se expresa a sí mismo si puede representarse utilizando los elementos definidos, sin utilizar aclaraciones extras.

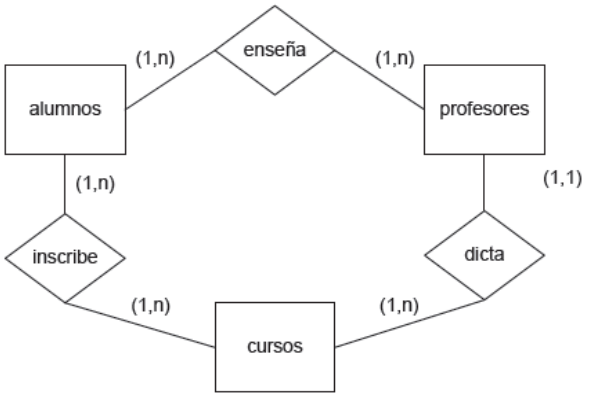
  Transformaciones para expresividad y autoexplicación:
  - Eliminar subentidades colgantes de la generalización (entidad hija que no tiene atributos)
  - Crear generalización entre entidades similares.
  - Dividir relaciones que puedan estar unidas innecesariamente
  - Crear subconjuntos

- **Completitud**: El modelo está completo cuando todas las características del problema están contempladas en él.
- **Corrección**: Un modelo es correcto si cada elemento en su construcción fue utilizado con propiedad (aspectos como la correctitud de las cardinalidades, identificadores y coberturas de las generalizaciones).
- **Extensible:** El modelo resulta extensible si es fácilmente modificable para incorporar nuevos conceptos.
- **Legibilidad:** El esquema es legible si la representación gráfica es adecuada.
- **Minimalidad:** Un esquema es mínimo cuando cada concepto se representa una sola vez en el modelo. Hay dos factores que pueden afectar la minimalidad:
  - Atributos derivados: Atributo que aparece en el modelo, pero cuya información, si no existiera almacenada en él, igualmente podría ser recuperada. **Si existen atributos derivados, el esquema no es mínimo**:
    - Pueden existir igualmente en ciertos casos para mayor facilidad.
    - La decisión puede pasarse al esquema lógico.



El atributo cantidad_de_materias_aprobadas se podría calcular contando la cantidad de relaciones entre alumnos y materias donde el atributo nota sea mayor a 4.

- Ciclos de relaciones: Existe un ciclo cuando una entidad A está relacionada con una entidad B, la cual está relacionada con una entidad C, la que a su vez se relaciona con la entidad A.
  No todos los ciclos afectan la minimalidad. Un ciclo afecta la minimalidad cuando una de las relaciones existentes puede ser quitada del esquema y el modelo sigue representando la misma información.



En este caso, si se quita la relación ‘inscribe’, el modelo sigue representado la misma información. Se conoce que profesores le enseñan a cada alumno. Al saber que un profesor solo puede dictar un curso, ya es posible determinar en que curso se encuentra inscripto el alumno utilizando la relación ‘enseña’.

Modelo lógico

Diseño lógico de alto nivel usando el modelo conceptual de entidad relación. Se convierte el esquema conceptual en uno lógico.

Objetivos

- Convertir el esquema conceptual en un modelo más cercano a la representación entendible por el SGBD (el cual debe definirse el tipo en este paso, **relacional**, OO, jerárquico o de red)
- Representar un esquema equivalente al conceptual, que sea más eficiente para su utilización

Entradas

- **Esquema conceptual:** resultado de la etapa anterior.
- **Descripción del modelo lógico a obtener:** se deben definir las reglas que se aplicarán en el proceso de conversión.
- **Criterios de rendimiento de la base de datos.**
- **Información de carga de la base de datos.**

Decisiones en el modelo lógico

➔ **Atributos derivados**: Ahora se debe tomar la decisión de dejarlos o no.
Ventajas:
- Disponibilidad de información rápida si es info. muy requerida.

Desventajas:
- Debe ser modificado cada vez que se modifica la información que lo contiene.

Es recomendable dejar aquellos que son utilizados frecuentemente.
Aquellos que necesiten mucho recálculo es recomendable quitarlos.

➔ **Ciclo de relaciones**: La decisión del analista pasa por tener el modelo mínimo, o por que posteriormente el modelo implique menos tiempo de procesamiento.

➔ **Atributos polivalentes**: En general, los gestores de bases de datos permiten que un atributo tenga múltiples valores, por solo en tamaño fijo. Por lo tanto, es difícil determinar de antemano un valor de espacio de almacenamiento. La **primera forma normal** indica que los atributos de entidades o relaciones **deben** ser simples. Como solución se plantea generar una nueva entidad que contenga los atributos y relacionarla con la entidad que tenía el atrib. polivalente.

➔ **Atributos compuestos**: Generalmente los gestores de bd no soportan estos atributos. Tres posibles soluciones
- ◆ Generar un atributo único concatenando los atributos simples del atributo complejo. Se pierde la identidad de cada atributo y es difícil de operar.
- ◆ Definir los atributos simples en la entidad. Mantiene la identidad de los atributos, pero aumentan la cantidad de los mismos en la entidad. Es generalmente la opción indicada.
- ◆ Generar una nueva entidad con los atributos del atributo compuesta. Opción más compleja.

➔ **Jerarquías**: Punto más importante de convertir del modelo conceptual al lógico. El modelo relacional no soporta el concepto de herencia. Tres opciones:
- ◆ Eliminar las especializaciones, dejando solo la entidad padre e incorporando todos los atributos de los hijos como opcionales
- ◆ Eliminar la generalización, dejar solo las especializaciones, incorporando los atributos del padre en cada uno de sus hijos.

- ◆ Dejar todas las entidades de la jerarquía y crear relaciones “ES_UN” uno a uno entre padre y cada uno de los hijos. Se mantienen los atributos originales y es la opción que mejor capta el concepto de herencia, pero se generan un mayor número de entidades y relaciones en el modelo lógico (posible problema de performance).

La cobertura de la jerarquía es la que más determina la solución viable:

- Si la cobertura es **Parcial**, la segunda solución no es viable, ya que rompe la equivalencia entre el modelo conceptual y el lógico, ya que se pierde información.
- Si la cobertura es **Superpuesta**, la segunda opción es aplicable pero poco práctica: si un elemento perteneciera a dos subentidades generadas se estaría repitiendo información.

➔ **Partición de entidades**: El motivo de partir una entidad es reorganizar la distribución de las entidades en un conjunto de entidades (**partición horizontal**) que la componen, o de los atributos (**partición vertical**) que conforman cada conjunto de entidades. El objetivo es mejorar la performance de la BD en las futuras operaciones.

- ◆ Horizontal: Permite separar un conjunto de entidades con múltiples atributos.
- ◆ Vertical: Analiza un conjunto de entidades con múltiples atributos. Agrupa distintos atributos en nuevas entidades. Por ejemplo: un empleado podría tener sus atributos divididos en tres conjuntos diferentes: datos personales, datos laborales, datos salariales.

Modelo físico

El modelo relacional físico representa a una base de datos como una colección de archivos denominados tablas, las cuales se conforman por registros. Cada tabla se denomina relación y está integrada por filas horizontales y columnas verticales.

- Cada fila representa un registro del archivo y se denomina **tupla**.
- Un atributo mantiene su nombre.
- El tipo de datos que describe los tipos de valores de un atributo se denomina **dominio**.

Pasos para la conversión del modelo lógico a físico

1) Eliminación de identificadores externos.
2) Selección de claves:
    a) Primaria
    b) Candidata
3) Conversión de las entidades
4) Conversión de las relaciones

Paso 1: Eliminación de identificadores externos

Cada una de las entidades que conforman el esquema lógico **deben** poseer sus identificadores definidos de manera interna. Para lograr esto, se deben incorporar dentro de la entidad que posea identificadores externos los atributos necesarios para identificar a esa entidad. En caso de existir más de un identificador, se deberá elegir el que luego será la clave primaria.

Paso 2: Selección de claves (primaria, candidata, secundaria)
El admin. de la BD debe decidir cuáles de los identificadores son claves primarias o candidata.

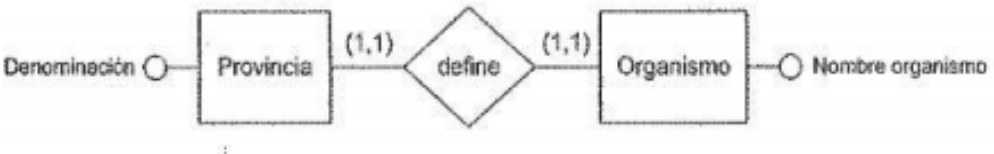
- ➔ Si una entidad tiene definido un único identificador, entonces es **clave primaria** de la tabla.
- ➔ Si una entidad tiene definidos varios identificadores, la selección debe realizarse de la siguiente manera:
  - ◆ Entre un identificador simple y uno compuesto, **debería tomarse el simple** dado que es más fácil de manejar.
  - ◆ Entre dos identificadores simples, se debe optar por aquel de **menor tamaño** físico.
  - ◆ Entre dos identificadores compuestos, se debería optar por aquel que tenga **menor tamaño en bytes.**

El resto de los identificadores serán definidos como **claves candidatas**.
El acceso físico al archivo de la tabla se realiza utilizando la **clave primaria,** el resto de las claves pueden utilizarse para generar índices secundarios que referencien al índice primario.
Actualmente los gestores de base de datos permiten el uso de un atributo del dominio **autoincremental** como clave primaria. El usuario solo puede consultarlo, no generarlo, borrarlo ni modificarlo. Es la opción más eficiente.

Concepto de superclave
Es un conjunto de uno o más atributos que permiten identificar de forma única una entidad de un conjunto de entidades. Presentada de esta manera, una superclave es similar a una clave primaria o candidata, sin embargo, **las superclaves pueden contener atributos innecesarios**. Si un atributo es superclave, entonces también lo es cualquier superconjunto que incluya a dicho atributo.

Paso 3: Conversión de entidades
En general, cada entidad definida en el modelo lógico es definida como tabla en el modelo físico. Al definir cada tabla, se considera el uso de los autoincrementales como claves primarias. En la conversión al modelo físico, **las claves candidatas no son indicadas**. El proceso de conversión muestra que todas las entidades deben convertirse en tablas, pero existe una excepción:

- Casos en el que existan dos entidades con una relación uno a uno. La existencia de una entidad determina la existencia de otra. Por lo tanto, en este caso es recomendable generar una única tabla con la información correspondiente de ambas entidades. Ejemplo:

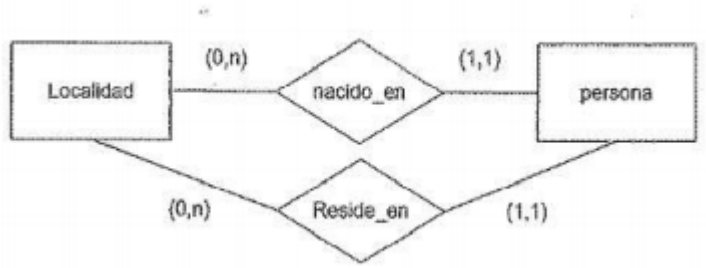


Paso 4: Conversión de las relaciones
Se deberán analizar tres situaciones diferentes, referidas a las cardinalidades de las relaciones.

➔ **Cardinalidad muchos a muchos**: La solución propuesta es independiente de la cardinalidad mínima definida (opcional u obligatoria). Las relaciones N a N se convierten en **tabla**. La tabla resultante contiene como atributos las claves primarias de las entidades relacionadas. Estos atributos, a su vez, generan la clave primaria de la tabla (en la práctica es más común que se utilice un autoincremental como CP que la clave primaria compuesta).

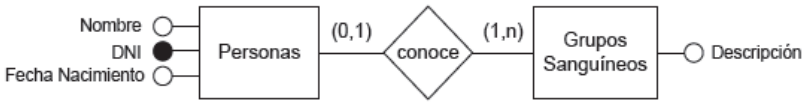
➔ **Cardinalidad uno a muchos**: Tiene dos posibles alternativas. La relación puede convertirse en tabla o no, dependiendo de la cardinalidad mínima definida y las decisiones de diseño

- ◆ Uno a muchos con **participación total**: Se agrega una clave foránea a la tabla correspondiente a la entidad cuya **cardinalidad máxima** de la relación es 1. No es necesario generar otra tabla. La clave foránea es clave secundaria en la tabla donde aparece.
- ◆ Uno a muchos con **participación parcial del lado de muchos**: Ejemplo



La solución es la misma que para el caso anterior. La tabla *PERSONA* tendría una clave foránea asociada a la localidad.

- ◆ Uno a muchos con **participación parcial del lado de uno:**



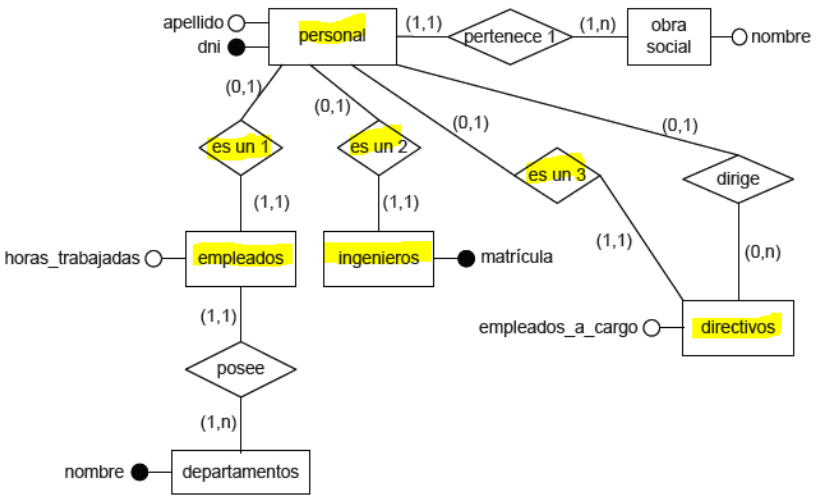
Se puede solucionar de dos maneras:

- Clave foránea que acepta valores nulos en la tabla cuya cardinalidad es (0, 1). **No es recomendado** usar valores nulos.
- Generar una tabla de la misma manera que se plantea en el caso de relaciones muchos a muchos.

- ◆ Uno a muchos con **cobertura parcial de ambos lados:** Se dispone de las dos soluciones planteadas para el ítem anterior.

➔ **Cardinalidad uno a uno**:

- ◆ Uno a uno **con participación total**: Dos posibles implementaciones
  - Agregar una clave foránea en alguna de las dos entidades (es indiferente en cual).
  - Unir ambas entidades en una, conteniendo todos los atributos.
- ◆ Uno a uno **con participación parcial de un lado**: La clave primaria de la entidad cuya participación es parcial se utiliza como clave foránea del lado de la entidad con participación obligatoria. Ejemplo:



EMPLEADOS = (idpersonal, horas_trabajadas)
INGENIEROS = (idpersonal, matricula)
DIRECTIVOS = (idpersonal, empleados_a_cargo)

- Uno a uno **con participación parcial de ambos lados**: No es un caso común. Se recomiendo generar una tabla como cardinalidad muchos a muchos.

Para las relaciones recursivas y ternarias, las conversiones también se realizan en función a las cardinalidades previamente nombradas.

Integridad referencial

Es una propiedad deseable de las bases de datos relacionales. Esta propiedad asegura que un valor que aparece para un atributo en una tabla, aparezca además en otra tabla para el mismo atributo. La integridad referencial plantea restricciones entre tablas y sirve para mantener la consistencia entre las tuplas de dichas tablas.

FACTURAS = (idfactura, fecha, monto, idcliente)
CLIENTES = (idcliente, nombre, dirección)

El atributo idcliente en la tabla FACTURAS es una CF. Esta CF permite establecer IR entre la tabla FACTURAS y la tabla CLIENTES. Para que exista IR entre dos tablas, necesariamente debe existir un atributo común (no es obligatorio que el atributo en común entre ambas tablas sea CP en una de ellas, podría haber excepciones).

Tipos de integridad referencial:

- **Restringir la operación**: Si se intenta borrar o modificar una tupla que tiene IR con otra, la operación se restringe y no se lleva a cabo.
- **Operación en cascada**: Si se intenta borrar o modificar una tupla sobre la tabla donde **está definida la clave primaria de la IR**, la operación se realiza en cadena sobre las tuplas donde se referencia esa tabla.
- **Establecer la clave foránea en nulo**: Si se borra o modifica el valor del atributo que es clave primaria, sobre la clave foránea se establece el valor nulo.

- **No hacer nada**: Se indica al gestor que no es necesario controlar la integridad referencial. Es equivalente a no definir restricciones de IR.

Tipos de restricciones

- **De dominio**: Especifican que el valor de cada atributo A debe ser un valor atómico del dominio de A.
- **De clave:** Evitar que el valor del atributo clave genere valores repetidos.
- **Sobre nulos**: Evita que un atributo tome el valor nulo en caso de no ingresarle ningún valor.
- **De integridad**: Ningún valor de la clave primaria puede ser nulo.
- **De integridad referencial:** Se especifica entre dos relaciones y sirve para mantener la consistencia entre las tuplas de esas dos relaciones. Establece que una tupla en una relación que haga referencia a otra relación deberá referirse a una tupla existente en esa relación. La clave foránea está representada por un atributo de una relación, donde ese mismo atributo es clave primaria en otra relación.

Violaciones a las restricciones

- **Alta**: Puede violar
    - Valor nulo para clave.
    - Repetición de clave
    - Integridad referencial.
    - Restricciones de dominio.
- **Baja**: Puede violar
    - Integridad referencial
- **Modificación:** Cualquiera de las anteriores.

Dependencias funcionales

Una dependencia funcional representa una restricción entre atributos de una tabla de la base de datos. Se dice que un atributo *Y* depende funcionalmente de un atributo *X* (denotado X → Y) cuando para un valor dado de X siempre se encuentra el mismo valor para el atributo Y. Generalizando, el atributo X determina el atributo Y.
El atributo X se lo denomina **determinante**, mientras que al atributo Y **consecuente.**
Cualquier atributo de una tabla **depende funcionalmente** de la clave primaria de esa tabla. Esto también es aplicable para el concepto de clave candidata. Cualquier atributo (incluida la clave primaria) depende funcionalmente de la o las claves candidatas de la tabla.

Conjunto mínimo de dependencias funcionales
Es fundamental definirlo. Este conjunto tiene las siguientes características:

- El consecuente de la DF debe estar formado por un **solo** atributo.
- Si se define la DF X → Y, no es posible encontrar otra DF Z → Y, donde Z es subconjunto de X. En este caso la DF se denomina **completa**.
- No se puede quitar de un conjunto de DF alguna de ellas y seguir teniendo un conjunto equivalente.

Tipos de dependencias funcionales

- ➔ Dependencia funcional **parcial**: Una DF X → Y es parcial cuando existe otra DF Z → Y, siendo Z un subconjunto de X. Este definición contradice la definición de conjunto mínimo (este caso trae aparejada repetición de información y, consecuentemente, anomalías de actualización). Este caso es muy **probable** que suceda en DF que tienen un determinante **compuesto.** Si el determinante es **simple**, no hay forma de que una DF parcial exista.
- ➔ Dependencia funcional **transitiva**: Una DF X → Y es transitiva cuando existe un atributo Z tal que X → Z y Z → Y. Nuevamente esta definición contradice la tercera propiedad del conjunto mínimo de DF, ya que se puede quitar del conjunto de DF alguna de ellas (por ejemplo, X → Y) y seguir teniendo un conjunto equivalente).
  Ejemplo: *ALUMNO* = (**idAlumno**, nombre, dirección, idCarrera, nombre_carrera)
  Se establecen las siguientes DFs:
  idAlumno → nombre
  idAlumno → dirección
  idAlumno → idCarrera
  idAlumno → nombre_carrera
  Se puede notar también que el nombre de la carrera también depende del id de la carrera:
  idCarrera → nombre_carrera
  La DF idAlumno → nombre_carrera genera una DF transitiva, que consecuentemente genera información repetida en la base de datos y otras anomalías de actualización.
- ➔ Dependencia funcional de **Boyce-Codd**: Se denomina dependencia funcional de Boyce-Codd cuando el determinante de la DF **no es una clave primaria ni candidata** y el consecuente **es una clave primaria, clave candidata o parte de ella**.

Normalización

Es un mecanismo que permite que un conjunto de tablas, que integran una BD, cumpla una serie de propiedades deseables. Lo que se busca evitar:

- Redundancia de datos: causa primaria de inconsistencias en la base de datos.
- Anomalías de actualización.
- Perdida de integridad de datos.

La normalización comienza examinando las relaciones existentes entre los atributos (**dependencias funcionales)**. Se busca identificar el agrupamiento óptimo de estos atributos.
El primer paso para un proceso de normalización es identificar la clave primaria y las candidatas de cada tabla del modelo.
El proceso de normalización es incremental, a medida que se va avanzando se vuelve más restrictivo. Se comienza con la BD en forma **no normal**.

Formas normales

- ➔ Primera forma normal: Un modelo está en 1FN si todos los atributos que conforman las tablas del modelo son atributos **monovalentes**. Este proceso se realiza sobre el

esquema **lógico**: luego, al obtener el esquema físico respectivo, todas las tablas estarán en 1FN

➔ Segunda forma normal: Un modelo está en 2FN si **está en 1FN** y además, no existe en ninguna tabla del modelo una **dependencia funcional parcial**. La solución para las DF parciales pueden ser:

- ◆ Quitar el atributo de la tabla, chequeando que no se pierda información.
- ◆ Los atributos que generan la DF parcial situarlos en una nueva tabla.

Por ejemplo, sea la tabla:
PEDIDOS = (idpedido, idproducto, descripciónproducto, fechapedido, cantidad)

Podemos identificar las siguientes DFs:
Idpedido, idproducto → descripcionproducto
Idpedido, idproducto → fechapedido
Idpedido, idproducto → cantidad

Se puede observar que la descripción del producto vendido es un atributo que depende del código del producto. Asimismo, la fecha del pedido depende funcionalmente del código del pedido. Por lo tanto se pueden definir dos DFs más que se definen como **DFs parciales:**

Idproducto → descripcionproducto
Idpedido → fechapedido

Para llevar esta tabla a 2FN es necesario mover los atributos que generen DFs parciales a nuevas tablas donde en ninguna se presenten DFs parciales:

PEDIDOS = (idpedido, fechapedido)
PRODUCTOS = (idproducto, descripciónproducto)
PEDIDOSPRODUCTOS = (idpedido, idproducto, cantidad)

➔ Tercera forma normal: Un modelo está en 3FN si **está en 2FN** y además, no existe en ninguna tabla una **dependencia funcional transitiva**. El procedimiento de normalización es el mismo, se deben quitar las DF transitivas de las tablas que las originan y mover los atributos a otra tabla (ya existente o no).

➔ Boyce-Codd forma normal: Un modelo está en forma normal Boyce-Codd (FNBC) si **está en 3FN** y además, no existe ninguna tabla del modelo con una **dependencia funcional de Boyce-Codd**.
Para generar un modelo de FNBC se **debe** garantizar que los determinantes de cada dependencia funcional sean siempre **claves candidatas**. La regla de normalización plantea que las dependencias funcionales de Boyce-Codd deben ser quitadas de la tabla donde se generan y llevarlas a una nueva tabla. La decisión de si es mejor detener el proceso de normalización en 3FN o seguir hasta BCFN depende de:

- La cantidad de redundancia que resulte de la presencia de una DF de Boyce-Codd.
- De la posibilidad de perder una clave candidata, que podría resultar perjudicial para realizar ciertos controles sobre los datos de la BD.

Ejemplo de dependencia funcional de BC y su proceso de normalización:
Se tiene una tabla donde se registra, para cada alumno, las materias en las que se inscribe y quién es el docente a cargo del dictado de cada una de ellas. Se sabe además que un docente solo dicta una materia.
DICTA = (nombre_alumno, nombre_materia, nombre_docente)

Existen dos CC para la tabla (nombre_alumno, nombre_materia) y (nombre_alumno, nombre_docente)

Se pueden observar las siguientes dependencias funcionales:
nombre_alumno, nombre_materia → nombre_docente
nombre_alumno, nombre_docente → nombre_materia

Pero además:
nombre_docente → nombre_materia

nombre_docente por sí solo **no es clave primaria ni clave candidata**, mientras que nombre_materia **es parte de una clave candidata**.

La solución que se plantea es generar una nueva tabla:
DICTA = (nombre_materia, nombre_docente)
CURSA = (nombre_alumno, nombre_materia)

Con estás dos tablas, se soluciona el problema de las DFs de BC, pero se pierde información, ya que no se puede determinar que docente le da clase a los alumnos. Por lo tanto, se debería agregar una nueva tabla.
CURSA_CON = (nombre_alumno, nombre_docente)

Dependencias multivaluadas
Una dependencia multivaluada denotada como X →→ Y, siendo X e Y conjunto de atributos en una tabla, indica que para un valor determinado de X, es posible determinar múltiples valores para el atributo Y. Una DM **no representa una situación anómala** para una BD. Con un valor de X se pueden obtener varios de Y.

El problema de la multideterminación de datos comienza cuando en una tabla cualquiera se tienen tres atributos, X, Y, Z, y se analizan las siguientes DM:

(X, Y) →→ Z
(X, Z) →→ Y

Pero, en realidad, tanto Y como Z solamente dependen de X en cada una de las DF. Sea el siguiente ejemplo:

SUCURSALES = (nombre_sucursal, propietario_sucursal, empleado_sucursal)

Nombre_sucursal	Propietario_sucursal	Empleado_sucursal
La Plata	Gomez	Bertagno
La Plata	Perez	Bertagno
La Plata	Gomez	Cappa
La Plata	Perez	Cappa

Se puede notar que en la tabla anterior fue necesario indicar dos veces que el empleado Bertagno trabaja en la sucursal La Plata, y dos veces que el propietario Gomez posee la sucursal La Plata. Esto se debe a que:

(Nombre_sucursal, propietario_sucursal) →→ Empleado_sucursal
(Nombre_sucursal, empleado_sucursal) →→ Propietario_sucursal

Tanto el empleado como el propietario dependen de la sucursal. Si, por ejemplo, se quisiera agregar un empleado a una sucursal, el empleado debería ser cargado dos veces ya que las sucursales tienen dos propietarios. Esta condición hace que se repita mucha información en la tabla.

- ➔ Cuarta forma normal: Un modelo está en 4FN si está en BCFN y, además, todas las **dependencias multivaluadas** que se puedan encontrar en las tablas son **triviales**.

  Así, es posible definir dos tablas con la siguiente estructura para el ejemplo anterior:

  EMPLEADOS_SUCURSAL = (nombre_sucursal, empleado_sucursal)
  PROPIETARIOS_SUCURSAL = (nombre_sucursal, propietario_sucursal)

  En ambos casos se mantienen DM, pero bajo la forma:

  Nombre_sucursal →→ empleado_sucursal
  Nombre_sucursal →→ propietario_sucursal

  Una DM *X* →→ *Y* se considera trivial si *Y* está multideterminado por el atributo *X* y no por un subconjunto de *X*.

  Dada la DM (*X*, *Z*) →→ *Y*, esta será trivial si *Y* está multideterminado únicamente por el par (*X*, *Z*). Si Y está multideterminado solamente por *X* o *Z*, la dependencia no se considera trivial.

- ➔ Quinta forma normal: Un modelo está en 5NF si y sólo sí está en 4NF, y no existen relaciones con dependencias de combinación. Una dependencia de combinación es una propiedad de la descomposición que garantiza que no se generen tuplas espurias al volver a combinar las relaciones mediante una operación del álgebra relacional. Ejemplo: En la siguiente tabla se muestra que doctor atiende para cada condición y el asegurador donde trabaja. Se puede ver en las filas amarillas que al agregar una nueva condición a atender hay información que se repite, por ejemplo que el doctor ‘Cárdenas’ atiende la condición ‘Depresión’.

Psiquiatra	Asegurador	Condición
Dr. Hernandez	Healthco	Ansiedad
Dr. Hernandez	Healthco	Depresión
Dr. Bran	FriendlyCare	OCD
Dr. Bran	FriendlyCare	Ansiedad
Dr. Bran	FriendlyCare	Depresión
Dr. Cárdenas	FriendlyCare	Esquizofrenia
Dr. Cárdenas	Healthco	Ansiedad
Dr. Cárdenas	Healthco	Demencia
Dr. Cárdenas	Victorian Life	Trastorno de conversión
Dr. Cárdenas	FriendlyCare	Depresión
Dr. Cárdenas	Healthco	Depresión
Dr. Cárdenas	Victorian Life	Depresión

Aplicando el proceso de normalización de 5FN:

QUINTA FORMA NORMAL

Psiquiatra	Condición
Dr. Hernandez	Ansiedad
Dr. Hernandez	Depresión
Dr. Bran	OCD
Dr. Bran	Ansiedad
Dr. Bran	Depresión
Dr. Cárdenas	Esquizofrenia
Dr. Cárdenas	Ansiedad
Dr. Cárdenas	Demencia
Dr. Cárdenas	Trastorno de conversión
Dr. Cárdenas	Depresión

Asegurador	Condición
Healthco	Ansiedad
Healthco	Depresión
Healthco	Demencia
FriendlyCare	OCD
FriendlyCare	Ansiedad
FriendlyCare	Depresión
FriendlyCare	Trastorno emocional
FriendlyCare	Esquizofrenia
Victorian Life	Trastorno de conversión

Psiquiatra	Asegurador
Dr. Hernandez	Healthco
Dr. Bran	FriendlyCare
Dr. Cárdenas	FriendlyCare
Dr. Cárdenas	Healthco
Dr. Cárdenas	Victorian Life

Dividiendo de está forma y haciendo los respectivos joins entre las tres tablas, se puede determinar que condiciones atiende cada doctor para cada una de las obras sociales donde trabaja, evitando generar información redundante sobre una única tabla.

Clase 3:
Lenguajes de consulta

Son aquellos lenguajes utilizados para operar con la base de datos. Se diferencian dos tipos:

- Procedurales: Instrucciones para realizar secuencia de operaciones (qué y cómo).

- No procedurales: Solicita directamente la información deseada (qué).

Álgebra relacional
Es un lenguaje de consulta procedural. Operaciones de una o dos relaciones (tablas) de entrada que generan una nueva relación como resultado.

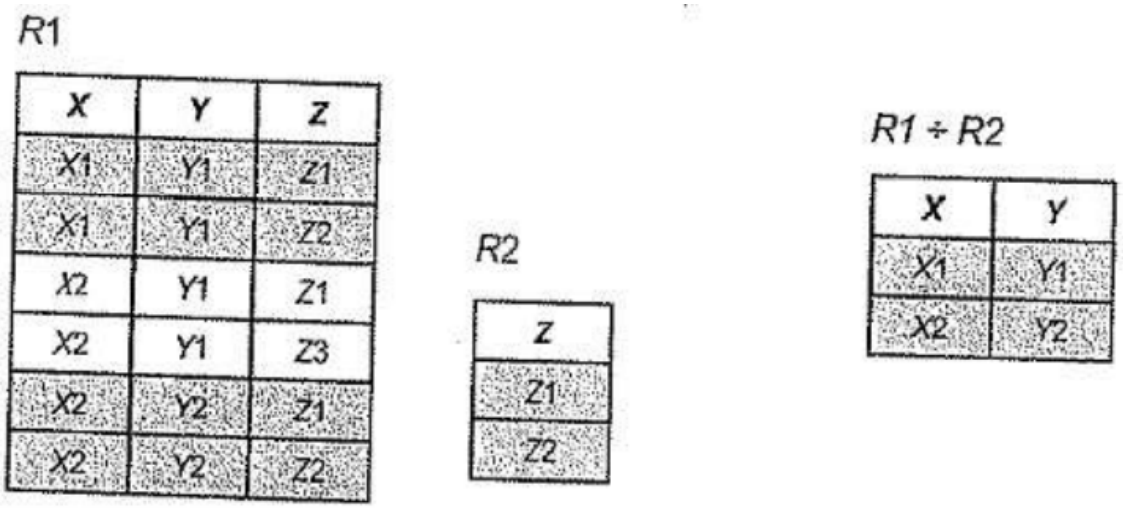
Operaciones fundamentales

➔ **Operadores unarios**: operan sobre una tabla o relación.
- ◆ Selección: permite filtrar tuplas de una tabla dado un predicado.
  Formato: σ_p**(Tabla)**
- ◆ Proyección: Permite presentar en el resultado algunos atributos de una tabla.
  Formato: $\pi_{atributos}$**(Tabla)**
  En el resultado se eliminan las tuplas repetidas.
- ◆ Renombre: **ρT(Tabla)**. Renombra la tabla "Tabla" a "T"

➔ **Operadores binarios**: operan sobre dos tablas o relaciones.
- ◆ Producto cartesiano: Es equivalente al producto cartesiano entre conjuntos. Se aplica a dos tablas o relaciones de la BD, y vincula cada tupla de una relación con **cada una** de las tuplas de la otra relación.
  Formato: **T1 × T2**
- ◆ Unión: Equivalente a la unión matemática de conjuntos. Se aplica a dos tablas o relaciones de la BD, generando una nueva tabla cuyo contenido es el contenido de cada una de las tablas involucradas en la operación.
  Formato: **T1** ∪ **T2**
  Las tuplas pertenecientes tanto a T1 como T2 no se duplican. T1 y T2 deben **ser compatibles** (sus esquemas deben ser **equivalentes en la cantidad**, **posición** y **dominio** de los atributos, aunque sus nombres sí pueden ser distintos).
- ◆ Diferencia: Equivalente a la diferencia matemática de conjuntos. Se aplica a dos tablas o relaciones de la BD, generando una nueva tabla donde se eliminan aquellas tuplas de la tabla T1 que también están presentes en la tabla T2.
  Formato: **T1 - T2**
  Ambas tablas deben ser **unión-compatible**.

➔ **Operadores adicionales**
- ◆ Producto natural: Reúne las tuplas de la primera tabla que se relacionan con la segunda tabla, descartando las tuplas no relacionadas. Ambas tablas deben poseer una atributo en común, sino la operación es similar al producto cartesiano. Se eliminan las columnas (atributos) repetidos.
  Formato: **T1 |×| T2**
- ◆ Intersección: Equivalente a la intersección matemática de conjuntos. Ambas tablas deben ser **unión-compatible**. Se aplica a dos tablas o relaciones de la BD, generando una nueva tabla con las tuplas comunes de ambas tablas.
  Formato: **T1** ∩ **T2**
- ◆ División: Dada dos tablas *R1* y *R2*, ***R1 % R2*** tiene como resultado los valores de atributos de R1 que se relacionan con todas las tuplas de R2. Esta operación unicamente es posible si y solo si **el esquema de R2 está incluido en el esquema de R1.** Es decir, todos los atributos de R2 son atributos de R1.

El formato de la tabla resultante es R1 - el esquema de R2.
Ejemplo:



- Asignación temporal: **A** ⇐ **Consulta**. Vuelca a A los resultados de CONSULTA. Luego es posible utilizar A.

Operaciones de actualización

- **Alta**

  Alumnos ⇐ Alumnos U {"Delia", "30777987", 2, 1}

- **Baja**

  Alumnos ⇐ Alumnos – $\sigma_{nombre = 'Lopez'}$ (alumnos)

- **Modificación**

  $\delta_{atributo = valor\_nuevo}$ (tabla)

  Ejemplo 15: Se debe modificar el número de DNI de Pettorutti por 34567231.

  $$\delta_{dni = '34567231'}(\sigma_{nombre = 'Pettorutti'} (alumnos))$$

Definición formal del álgebra relacional

Una expresión básica de en álg. relacional consta de:

- Una relación o tabla de la BD.
- Una relación o tabla constante (ejemplo de la inserción, donde la tupla es expresada literalmente)

Una expresión (consulta) general E en álg. relacional se construye a partir de subexpresiones (subconsultas) E_1, E_2, E_N

Las expresiones posibles son:

- σ_p (E1) P Predicado con atributos en E1
- π_S(E1) S Lista de atributos de E1
- ρ_Z(E1) Z Nuevo nombre de E1
- E1 × E2
- E1 ∪ E2

- E1 - E2

Clase 4:
SQL: Lenguaje de consultas estructurado

Es un lenguaje de consultas de base de datos, compuesto por **dos módulos fundamentales**:

Módulo para definición del modelo de datos, definido como **DDL**, con el cual se pueden definir las tablas, atributos, integridad referencial, índices y restricciones del modelo.

Crear tabla

```
CREATE TABLE Empresa (
        idempresa INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
        nombre_empresa VARCHAR(100) NOT NULL,
        cuit VARCHAR(10) NULL,
        PRIMARY KEY(idempresa)
        UNIQUE INDEX empresa_index(nombre_empresa)
);
```

Modificación de tabla

```
ALTER TABLE Empresa (
        ADD COLUMN razon_social VARCHAR(100) NULL,
        DROP COLUMN cuit,
        ALTER COLUMN nombre_empresa VARCHAR(150)
);
```

Baja de tabla

```
DROP TABLE Empresa;
```

Módulo para la operatoria normal de la base de datos, denominado **DML**, con cual se pueden realizar las consultas, altas, bajas, y modificaciones de los datos sobre las tablas del modelo.

Operadores:

- SELECT
- INSERT
- UPDATE
- DELETE

Estructura básica:
SELECT atributos
FROM tabla/s
WHERE predicado

En la Cláusula **WHERE** pueden usarse operadores lógicos como AND, OR, NOT, etc. Algunos operadores comunes:

- BETWEEN *valor_1* AND *valor_2*: Verifica si el valor de un atributo está contemplado dentro del rango de valores especificados.
- IS NULL / IS NOT NULL: Verifica que el valor del atributo sea o no null.

- LIKE: Utilizado para validar atributos del tipo cadena/string.

En la cláusula **FROM** se indican las tablas en las cuales se va a realizar la consulta. Se puede renombrar las tablas indicando un nuevo nombre posterior al nombre original (*FROM Empresa e, Empleado emp*).

JOINS:

- Producto cartesiano, se listan las tablas separados por coma.
- Producto natural: Tabla1 INNER JOIN Tabla2 ON (condición de igualdad entre los atributos de ambas tablas). **Repite** atributos de ambas tablas. Es necesario especificar el atributo en común de ambas tablas.
- Producto natural por equi combinación de CF y CP: Tabla1 NATURAL JOIN Tabla2. **No repite** atributos de ambas tablas.
- LEFT JOIN: Producto natural, pero se conservan todos los registros de la tabla de la izquierda, completando con NULL los casos de no correspondencia.
- RIGHT JOIN: Inversa del anterior

Operadores para atributos

- SELECT *: se muestran todos los atributos de las tablas indicadas en el FROM
- DISTINCT: elimina las tuplas repetidas
- AS para renombrar atributos

Cláusula ORDER BY

Especifica el atributo por el cual las tuplas van a ser ordenadas. Dos alternativas: ASC (default) o DESC.

Operaciones sobre conjuntos

- UNION: Agrupa las tuplas resultantes de dos subconsultas
- UNION ALL: Similar al anterior, pero conservando todos los duplicados
- INTERSECT: Intersección entre dos subconsultas
- EXCEPT: Diferencia entre dos subconsultas

Funciones de agregación

Funciones que operan sobre un conjunto de tuplas y producen un único valor de salida. No pueden estar definidas dentro de una cláusula **WHERE.**

- AVG: Aplicable a atributos numéricos, retorna el promedio de un atributo.
- MIN: Retorna el valor minimo para un atributo dentro de las tuplas de la tabla.
- MAX: Retorna el valor máximo para un atributo dentro de las tuplas de la tabla.
- SUM: Aplicable a atributos numéricos, suma el valor de un atributo para todas las tuplas de la tabla.
- COUNT: Cuenta las tuplas resultantes.

Funciones de agrupamiento

Permite agrupar un conjunto de tuplas por algún criterio

- HAVING: Permite aplicar condiciones a los grupos. Dentro de esta cláusula se **pueden usar funciones de agregación**
- GROUP BY *atributo*: genera N subconjunto de tuplas, cada uno agrupado por *atributo* diferente.

Ejemplo: Mostrar la cantidad de clientes por cada país.
SELECT COUNT(clienteID), pais
FROM Cliente
GROUP BY pais;

Operadores entre subconsultas

- = Cuando una subconsulta retorna un único resultado, es posible compararlo contra un valor simple.
- IN / NOT IN: Comprueba si un elemento está o no dentro de un conjunto.
- *valor OPERADOR* (<, >, >=, <=, <>) SOME(*subconsulta*): Compara *valor* contra cada valor resultante de la subconsulta. Retorna verdadero si existe por lo menos una valor dentro de la subconsulta que cumpla con la condición.
- *valor OPERADOR* (<, >, >=, <=, <>) ALL(*subconsulta*): Similar al anterior, pero para que retorne verdadero, todos los elementos de la subconsulta deben cumplir la condición del operador.
- EXISTS: Verdadero si la subconsulta contiene al menos una tupla resultante.

Vistas

Las vistas tienen la misma estructura que una tabla: tuplas y atributos. Son el resultado de guardar en forma temporaria una consulta SQL. Solo se almacena la definición de la consulta, no los datos. Cada vez que la vista es utilizada, la consulta almacenada es recalculada.

Clase 5:
Optimización de consultas

Los gestores de bases de datos presentan en general un optimizador de consultas. Este proceso se encarga de encontrar una consulta equivalente a la generada por el usuario que sea óptima en términos de performance. El proceso de optimización comienza con la consulta generada por el usuario, aplicando los siguientes pasos posteriormente:

- Un **parser** (analizador sintáctico) genera una expresión manipulable por el optimizador de consultas. Se convierte el texto de entrada (la consulta) en una estructura de tipo árbol, que resulta más útil para su análisis.
- A partir de la expresión generada, el optimizador obtiene una consulta equivalente más eficiente
- Por último, el proceso de optimización considera el estado actual de la BD y los índices definidos, para resolver la consulta que se tiene hasta el momento, con el acceso a disco posible.

Costo

Se pueden identificar los componentes del costo de ejecución de una consulta:

- **Costo de acceso** a almacenamiento secundario: acceder al bloque de datos que reside en disco.
- **Costo de computo**: costo de realizar operaciones sobre memoria RAM.
- Si es un sistema distribuido, también se debe tener en cuenta el **costo de comunicación**.

Optimización lógica
Existe una secuencia de resolución. Se pueden encontrar expresiones que sean más eficientes que otras.

Operación de selección
Se debe realizar la operación de selección lo antes posible en una consulta.
Ejemplo:

$$\pi_{dni}(\sigma_{localidades.nombre = \text{"La Plata"}}(alumnos \,|\times|\, localidades))$$

$$\pi_{dni}(alumnos \,|\times|\, (\sigma_{localidades.nombre = \text{"La Plata"}}(localidades)))$$

La segunda consulta es mejor en términos de performance, ya que solo se realiza el join con aquellas tuplas ya filtradas por el nombre de la localidad.

Operación de proyección
Esta operación también debe realizarse lo antes posible. Aplicando una proyección temprana, las tuplas resultantes son de menor tamaño, por lo que cada tupla también ocupará menor espacio en el buffer de memoria.

Operación de producto natural
Expresiones que realicen el producto natural entre **más** de dos tablas (T1 |x| T2 |x| T3) deben intentar resolverse en pasos:

- Primero realizar el producto natural T1 |x| T2
- Luego, aplicar el producto natural con T3 con las tuplas resultantes.

Además, no es lo mismo T1 |x| T2 que T2 |x| T1 desde un punto de vista de performance. El resultado es similar, pero se tienen distintos tiempos de respuestas. Si las tablas no tienen atributos en común, ambas consultas son similares.

- **Si una de las tablas tiene el atributo en común como CP, conviene realizar la operación entre la tabla con clave foránea sobre la tabla con CP.** (T_{CF} |X| T_{CP})
  Cada tupla de T_{CF} se reune con **una única** tupla de T_{CP}, además la búsqueda en T_{CP} se realiza por **clave primaria,** que es la manera óptima de hacerlo.
- Si hay un atributo en común pero **no es clave primaria en ninguna de las tablas**, entonces el producto debe realizarse sobre aquella tabla que **tenga mayor cantidad de elementos diferentes (mayor CV) para el atributo en común**. Para poder resolver estas situaciones, el optimizador dispone de estadísticas de la base de datos (cantidad de tuplas de las tablas, distribución de valores para los atributos). Estas estadísticas se mantienen parcialmente actualizadas (tienen alto coste de actualización).

Valores asociados a la optimización de consultas

- ➔ CT tabla: cantidad de tuplas de la tabla
- ➔ CB tabla: cantidad de bytes que ocupa cada tupla de la tabla
- ➔ CV(a, tabla): cantidad de valores diferentes del atributo a en la tabla.
- ➔ Costo **en bytes** selección:
  - ◆ $(CT_{TABLA} / CV(atributo, TABLA)) * CB_{TABLA}$
- ➔ Costo **en bytes** de proyección:

- ◆ $(CB_{ATRIBUTO\ 1} + CB_{ATRIBUTO\ 2} + CB_{ATRIBUTO\ N}) * CT_{TABLA}$

➔ Costo **en bytes** producto cartesiano:

- ◆ $(CT_{TABLA\ 1} * CB_{TABLA\ 1}) + (CT_{TABLA\ 2} * CB_{TABLA\ 2})$

➔ Costo de producto natural con atributo “a” en común

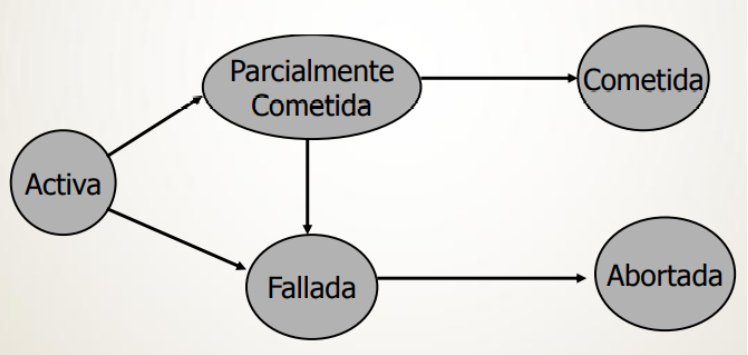
- ◆ $(CT_{TABLA\ 1} * CT_{TABLA\ 2}) / (MAX(CV(a, TABLA\ 1)), CV(a, TABLA\ 2)))$

Clase 6:
Transacciones

Una transacción es una colección de operaciones que forman una única unidad lógica de trabajo. Una transacción se **completa** cuando se ejecutaron **todas** las instrucciones que la componen. En caso de no poder ejecutar todas las instrucciones, ninguna de ellas debe llevarse a cabo. Para garantizar que una transacción mantenga la consistencia de la base de datos, es necesario que se cumpla con cuatro propiedades básicas (**ACID**):

1) **Atomicidad**: Garantiza que se ejecutan todas las operaciones sobre la BD, o ninguna de ellas se lleva a cabo. Responsabilidad del gestor de BD.
2) **Consistencia**: La ejecución de la transacción parte de una BD consistente y finaliza también con la BD consistente. Responsabilidad del encargado de escribir la transacción.
3) **Aislamiento (Isolation)**: Cada transacción actúa como única en el sistema (no debe afectar a la ejecución de otra transacción).
4) **Durabilidad**: Una vez finalizada la transacción, los efectos producidos en la BD son permanentes.

Estados de una transacción



- **Activa**: Estado inicial, estado normal durante la ejecución. Se mantiene hasta completar la última instrucción, o hasta que se produzca un fallo.
- **Parcialmente cometida/finalizada:** Se alcanza en el momento posterior a que se ejecuta la última operación de la transacción.
- **Cometida**: Se obtiene cuando la transacción finalizó su ejecución y sus acciones fueron correctamente almacenadas en memoria secundaria.
- **Fallada:** Cuando la transacción no puede continuar con su ejecución normal.
- **Abortado**: Este estado garantiza que una transacción fallada no ha producido ningún cambio en la BD.

Fallos

Se pueden diferenciar dos tipos de fallos:

- **Fallos que no producen pérdida de información**: Por ejemplo, transacciones solo de consulta.
- **Fallos que producen pérdida de información:** Por ejemplo, las transacciones que incluyen cláusulas SQL con INSERT, DELETE, UPDATE. Afectan al contenido de la BD y debe asegurarse que la integridad de datos se mantenga.

Métodos de recuperación de integridad de una base de datos: Atomicidad
Estos métodos ponen especial énfasis en tratar la información contenida en memoria RAM y el disco rígido. A priori, se pueden observar dos posibilidades:

- Volver a ejecutar completamente la transacción fallida: Al empezar desde una BD inconsistente, el resultado también lo será. Esta solución no sirve.
- No hacer nada: La BD permanecerá inconsistente. Tampoco sirve.

Para evitar los problemas que puedan surgir, es necesario realizar acciones que permitan **retrotraer el estado de la BD al instante anterior al comienzo de ejecución de la transacción.**
Existen dos métodos de recuperación: el de **Bitácora (log)** y **Doble paginación**. Ambos métodos necesita de dos algoritmos básicos:

- Algoritmos que se ejecuten durante el procesamiento de la transacción, que realizan acciones que permiten recuperar la consistencia de la BD ante un fallo.
- Algoritmos que se activan luego de detectar un error durante la transacción, que permiten recuperar el estado de consistencia de la BD.

Registro histórico / Bitácora

Todas las acciones llevadas a cabo sobre la DB deben quedar registradas en un archivo histórico de movimientos. Todos los movimientos producidos por las transacciones se registran en un archivo auxiliar y externo a la BD, **antes de producir los cambios sobre la BD.** Si se produce un error, se tiene constancia de todos los movimientos efectuados.
La estructura del archivo de bitácora es simple: cada transacción debe indicar su comienzo, finalización y cada una de las acciones llevadas a cabo sobre la BD **que modifiquen su contenido.** El gestor de la BD se encarga de:

- Controlar la ejecución de las transacciones.
- Administrar el archivo de bitácora
- Utilizar algoritmos de recuperación ante fallos.

Entradas:

- <T iniciada>
  - La entrada comienza su ejecución
- <T, E, Va, Vn> (*Operación de escritura*)
  - T: Identificador de la transacción
  - E: Identificador del elemento de datos
  - Va: Valor anterior
  - Vn: Valor nuevo
- <T Commit>
  - Transacción T terminó sin fallos
- <T Abort>

- Transacción abortada

Si una transacción registra un inicio pero no un fin (por un fallo como un corte abrupto de suministro), debe abortarse. Se **debe garantizar** que primero se graben los buffers del archivo de bitácora en memoria secundaria, antes de grabar en disco los buffers de la BD. La bitácora presenta dos alternativas para implementar el método de recuperación:

1) **Modificación diferida de la BD**: Las operaciones *WRITE* se aplazan hasta que la transacción esté parcialmente cometida. En ese momento se actualiza la bitácora y la BD.
   Ventajas:
   - Si la transacción falla, ningún cambio tuvo impacto sobre la BD, el fallo es ignorado y la BD permanece consistente.
   - No se necesita guardar en la bitácora el valor viejo del dato.

   Caso a tener en cuenta:
   Si se produce un fallo **luego de escribir** en la bitácora, durante la actualización efectiva de la BD, se puede perder la integridad. Cuando se vuelve del fallo, el algoritmo de recuperación **ignoraría** la transacción, ya que la misma tiene su inicio y su commit en la bitácora. Sin embargo, se activa el algoritmo **REDO**, que vuelve a ejecutar todas las transacciones que en bitácora tengan un start y un commit, devolviendo la integridad. Este algoritmo se ejecuta, **incluso si la BD esta en estado de consistencia.**
   Las transacciones tienen una condición llamada **idempotencia**, que significa que aunque se reejecute N veces una transacción, siempre se genera el mismo resultado sobre la BD

2) **Modificación inmediata de la BD**: La actualización de la BD se realiza mientras la transacción está activa y se va ejecutando. Reducen la sobrecarga de trabajo al pasar muchas transacciones al disco a la vez. Ante un fallo, la BD puede estar parcialmente modificada, por lo que se necesita además un algoritmo **UNDO**, que retrotrae la BD al estado que tenia antes de comenzar la transacción.En este caso, **es necesario** que se guarden los valores viejos de cada dato modificado en la bitácora.
   Entonces ante un fallo, y luego de recuperarse:
   - REDO(Ti), para toda *Ti* que tenga un Start y un Commit en la Bitácora.
   - UNDO(Ti), para toda *Ti* que tenga un Start y no un Commit.

Buffers de bitácora
Grabar en disco cada registro de bitácora insume gran costo de tiempo, por lo tanto se utilizan buffers. Una transacción está parcialmente cometida **después de grabar en memoria no volátil el commit en la bitácora**, esto implica que todos los registros anteriores de esa transacción ya están en memoria no volátil.

Puntos de verificación
Cuando ocurre un fallo, sin importar el tipo de modificación, todas las transacciones que tengan un start y un commit deberían rehacerse (**REDO**), lo que sería un trabajo innecesario, ya que la gran mayoría no habrían presentado fallos.

Para evitar la revisión de toda la bitácora ante un fallo, el gestor agrega en forma periódica una entrada **check-point**. Se agregan cuando hay seguridad de que la BD **está consistente**. Ahora ante un fallo, sólo debe revisarse la bitácora desde el punto de verificación en adelante.
Si los checkpoints se colocan muy cercanos, se rehace poco trabajo en caso de que se detecte un fallo, pero hay mas carga de escritura en la bitácora.
Si los checkpoints se colocan muy lejanos, hay menos carga de escritura en la bitácora, pero en caso de presentarse un fallo, se deben rehacer mayor cantidad operaciones.

Doble paginación

Este método planea dividir la BD en nodos virtuales (páginas) que contienen determinados datos. Se generan dos tablas **en disco**, y cada una de las tablas direcciona a los nodos (páginas) generados. Ante una transacción que realiza una operación de escritura sobre la BD, la secuencia de pasos es:

1) Se obtiene desde la BD el nodo (página) sobre el cual debe realizarse la escritura.
2) Se modifica el dato en memoria principal y se graba en disco, en un nuevo nodo, la página actual que referencia al nuevo nodo.
3) Si la operación finaliza correctamente, se modifica la referencia de la pagina a la sombra para que apunte al nuevo nodo.
4) Se libera el nodo viejo.

Ventaja: Menos accesos a disco
Desventaja: Complicada en un ambiente concurrente/distribuido.

Fallos
Luego de la recuperación, se copia la tabla de páginas sombra en memoria principal. Abort automáticos, se tiene la dirección de la página anterior sin las modificaciones.

Ventajas sobre el método de bitácora:

- Se elimina la sobrecarga por las escrituras al registro histórico
- La recuperación ante un fallo es mucho más rápida: Se recupera la pagina sombra como página actual, descartando la página actual)

Desventajas:

- Exige dividir la BD en nodos o páginas.
- Fragmentación de datos: Cambia la ubicación de los datos continuamente.
- Ante un fallo queda una página que no es más referenciada: se necesitan algoritmos que funcionen como garbage-collectors.

Entornos concurrentes

Los SGBD, a través de los gestores de BD, permiten que varias transacciones operen simultáneamente sobre la BD, situación que puede generar problemas de consistencia.
Dos transacciones, T1 y T2, son consistentes ejecutadas en entornos monousuarios, pero pueden no serlo si se ejecutan en simultáneo (ocurre **por la propiedad de aislamiento**).

Una solución podría ser secuencializar la ejecución de las transacciones ejecutando las mismas como monousuario, pero no es factible para entornos donde se tienen una gran cantidad de usuarios.

Planificación
Es necesario establecer un criterio de ejecución para que la concurrencia no afecte la integridad de la BD.
En un entorno concurrente, el orden de ejecución en serie de transacciones se llama planificación. Involucra **todas las instrucciones de las transacciones y se conserva el orden de ejecución** de las misma. Cada transacción debe ser analizada como un todo, sin quitarle instrucciones. Se garantiza la secuencialidad, pero no aprovecha el entorno concurrente.
Un entorno concurrente con N transacciones en ejecución **puede** generar N! trazas o planificadores en serie. La ejecución concurrente no necesita una planificación en serie. Cuando la ejecución de una transacción no afecta el desempeño de otra transacción sobre la BD, ambas pueden ejecutarse concurrentemente sin afectar la propiedad de aislamiento.

Conclusiones
- La inconsistencia temporal puede ser causa de inconsistencia en planificadores en paralelo.
- Una planificación concurrente debe **equivaler** a una planificación en serie, para que se mantenga la integridad de la BD
- Solo las instrucciones READ y WRITE deben considerarse.

Sea INS1 perteneciente a la transacción T1,
Sea INS2 perteneciente a la transacción T2,
Si INS1 e INS2 operan sobre datos diferentes**, no hay conflicto.**
Si INS1 e INS2 operan sobre los mismos datos, si cualquiera de la operaciones es un **WRITE,** están en conflicto. Si solo son operaciones **READ,** no hay conflicto.

Una planificación **P1** se transforma en una planificación **P2** mediante intercambios de instrucciones no conflictivas de **P1**, entonces P1 y P2 son **equivalentes en cuanto a conflictos**.
Esto significa que si:
- P2 es consistente, P1 también lo será
- P2 es inconsistente, P1 también será inconsistente

P2 es serializable en cuanto a conflictos si existe otra planificación **P1** equivalente en cuanto a conflictos con **P2**, y además, **P1** es una planificación en serie.

Métodos de control de concurrencia

Bloqueo
Se basa en que una transacción debe solicitar el dato con el cual desea operar y tenerlo en poder hasta que no lo necesite más. Similar al bloqueo que realiza el sistema operativo. En el caso de la transacciones, la BD actúa como un recurso compartido. Cada transacción debe bloquear la información que necesita, utilizarla y liberarla. Existen dos **tipos de bloqueos**:

- Compartido: Debe ser realizado por una transacción para leer un dato que **no modificará**. Mientras dure este bloqueo, se impide que otra transacción pueda **escribir** ese mismo dato. Un bloqueo compartido puede coexistir con otro bloqueo compartido sobre el mismo dato.
- Exclusivo: Se genera cuando una transacción necesita escribir un dato. Se bloquea la operación de **lectura y escritura** para ese dato. Un bloqueo exclusivo es único.

Si una transacción solicita un dato que está siendo utilizado, la transacción tiene dos posibilidades:
- Fallar y abortar, para luego comenzar como una transacción diferente.
- Esperar a que se libere al dato.

Aún utilizando protocolos de bloqueo, no se garantiza el aislamiento de una transacción. La solución que se plantea es que una transacción pida todos los recursos que necesita al principio de la misma y al finalizar los libere todos juntos.
Trasladar los bloqueos al principio de la transacción no soluciona el problema al 100%, además, se pueden dar situaciones de **deadlock**.
Entonces tenemos:
- Si se llevan los bloqueos al principio de la transacción, se puede llegar a una condición de deadlock.
- Si se realizan los bloqueos en el momento necesario, se puede dar una situación de perdida de consistencia.

Es **preferible** generar situaciones de deadlock por sobre inconsistencias.

Para asegurar integridad, el protocolo de bloqueo usa regla bien definidas: **protocolo de dos fases**. Este protocolo garantiza el aislamiento en la ejecución de las transacciones utilizando el principio de las dos fases:
- Fase de crecimiento: se solicitan bloqueos similares o se crece de compartido a exclusivo.
- Fase de decrecimiento: exclusivo, compartido, liberar datos. Baja el grado de restricción del bloqueo.

Este protocolo **garantiza** seriabilidad (por lo tanto, **aislamiento**), pero no evita situaciones de deadlock.

Protocolo basado en hora de entrada
Es una variante del protocolo de bloqueo, donde la ejecución exitosa de una transacción se establece de antemano, en el momento en que la transacción fue generada. Cada transacción recibe una **hora de entrada (HDE)** en el momento en que inica su ejecución. La HDE es una marca única asignada a cada transacción y puede representar el valor de un contador o la hora interna del servidor de la BD.
El método continúa asignando a cada elemento de dato de la BD dos marcas temporales:
- La hora de última lectura **HL(D)**
- La hora de última escritura **HE(D)**

Estas marcas corresponden a la HDE de la última transacción que leyó o escribió el dato.

Operación de lectura

- T1 desea leer el dato D
- Para poder leer el dato, se debe cumplir que HDE(T1) > HE(D). Esto asegura que cualquier transacción que intente leer una dato **debe haberse iniciado después** de la última escritura del dato.
- Si no se cumple la condición anterior, el dato D es demasiado nuevo para que T1 pueda leerlo. Una transacción posterior a T1 pudo escribir D. Si T1 llegará a leer ese dato, se podría perder la consistencia.

Operación de escritura

- T1 necesita escribir el dato D
- Si HDE(T1) < HL(D), la operación falla, ya que D fue leído por otra transacción posterior a T1.
- Si HDE(T1) < HE(D) la operación falla, ya que D fue escrito por una transacción posterior a T1.
- En cualquier otro caso, T1 puede ejecutar la escritura y establecer el HE(D) = HDE(T1)

Si la transacción falla en cualquiera de los dos casos, se aborta y entonces es posible generar una nueva transacción con un nuevo valor de HDE.

Granularidad

El concepto de granularidad en el acceso a los datos está vinculado con el tipo de bloqueo que se puede realizar sobre la BD

- La granularidad gruesa permite solamente **bloquear toda la BD**
- A medida que la granularidad disminuye, es posible bloquear a nivel tabla, registro o hasta campos que la componen

En general, los SGBD permiten diversos niveles de bloqueo sobre los datos. El nivel mayor de bloqueo es sobre la BD completa, está normalmente reservada para el diseñador de la BD. Las transacciones en entornos concurrentes necesitan bloqueos a nivel de registros (tuplas).

Bitácora en entornos concurrentes

Se aplica casi sin cambios en entornos concurrentes. La finalidad de este protocolo es garantizar la atomicidad de la transacción ante posibles fallos originados en su ejecución. Se agrega un nuevo tipo de fallo: una transacción que no puede continuar su ejecución por problemas de bloqueo o acceso a la BD. Como en cualquier otra situación de fallo, se debe retrotraer la BD a un estado consistente.
Consideraciones adicionales en entornos concurrentes:

- Único buffer de datos tanto para la BD como para la bitácora.
- Cada transacción tiene su propia área donde se administra la información localmente.
- El fallo de una transacción implica deshacer el trabajo llevado a cabo por ella.

Retroceso en cascada

El retroceso en una transacción puede llevar a que otras también fallen. Por este motivo se agrega una nueva condición. **Una transacción T_J no puede finalizar su ejecución si una transacción T_I anterior (que utiliza datos que T_J necesita) no finalizó su ejecución**.
De esta forma si T_I falla, T_J también deberá fallar.

Puntos de verificación (checkpoints)

El único cambio que requiere en ambientes concurrentes con respecto a monousuarios está vinculado con los puntos de verificación.

En entorno monousuarios, los checkpoints se agregaban periódicamente luego de <T_I finaliza> y antes de <T_{i+1} comienza>

En un esquema concurrente, no se puede garantizar la existencia en un momento temporal, donde ninguna transacción se encuentre activa.

La sentencia **<punto de verificación (L)>** agrega un parametro L, que contiene la lista de transacciones que, en el momento de agregar el checkpoint, se encuentran activas.