.UBAfiuba
FACULTAD DE INGENIERÍA

Apuntes de Bases de datos

[7515] Base de Datos
2C 2021

Grassano, Bruno
bgrassano@fi.uba.ar

Índice

1. Introducción a bases de datos **4**

2. Bases de datos tradicionales vs. no tradicionales **4**

3. Sistemas de gestión de bases de datos **4**

4. Arquitectura de 3 capas ANSI/SPARC **5**

5. Practica: Ejercicio AFA y Taller I **13**

6. Modelo relacional **16**

7. Practica: Modelo relacional **18**

8. Álgebra relacional **21**

9. Calculo Relacional **27**

10.Practica: Álgebra relacional **29**

11.SQL **34**

12.Practica: Taller SQL **40**

13.Practica: Taller Valores Nulos **50**

14.Teoría del Diseño Relacional **52**

15.Forma Normal **52**

16.Practica: Normalización **61**

17.Practica: Normalización II **63**

18.Concurrencia y Transacciones **65**

19.Bases de datos espaciales **72**

20.NoSQL **75**

21.Practica: MongoDB **82**

22.Practica: Agregación en MongoDB **85**

23.Recuperabilidad **90**

24.Seguridad **96**

25.Practica: Neo4J **99**

26.Procesamiento y Optimización de consultas **103**

27.Practica: Costos **108**

28.Data warehousing **109**

29.Practica: Resolución de consultas **111**

Introducción

El presente archivo contiene los apuntes que fueron tomados a lo largo de la cursada de la materia Base de Datos (75.15).

La materia esta muy buena, sin embargo hay que tener en cuenta que es pesada debido a la gran cantidad de temas que abarca.

Muy recomendable ir siguiendo la materia al día e ir haciendo los parcialitos (fueron 6 este cuatri) para llegar bien al parcial. Después para el final también recomiendo seguirla al día, ya que se puede evaluar cualquier tema de los vistos (ya sea teórico o practico)

Clase 1

1. Introducción a bases de datos

Palabras clave: lugar, datos, persistente, datos interrelacionados (que provienen de un cierto universo en común, tienen relaciones)

Definición: Una base de datos es un conjunto de datos interrelacionados.

Definición: Un dato es un hecho que puede ser representado y almacenado de alguna forma, y que tiene un sentido implícito. Lo puedo codificar de alguna forma, no es algo abstracto

---

2. Bases de datos tradicionales vs. no tradicionales

Definición: El predicado es una función que toma uno o mas argumentos y devuelve un valor de verdad.

Ejemplos

- La mesa 5 consumió 2 milanesas napolitanas y 1 botella de vino
- 100 gramos de chocolote poseen 546 calorias

Todas las oraciones van a tener un sujeto, verbo y predicado.

Las bases de datos almacenan proposiciones

- Juan gano x en 2012
- Martin gano x en 2015

Todas tienen la misma estructura, se pueden tipificar en un predicado al tener la misma estructura. Podemos separar el nombre, el x, y el año.

Puedo definir una función, que tome como variable las cosas en común que identificamos antes y devuelva Verdadero o Falso de acuerdo a la entrada. Las bases de datos solo almacenan proposiciones verdaderas. Toma determinados predicados, y guarda aquellos que son verdaderos. Estas son las bases de datos **tradicionales**.

Actualmente las bases de datos pueden almacenar datos mas complejos ahora. *Imágenes, audio, vídeos, etc* Estas son del tipo base de datos **no tradicional**.

Lectura recomendada: What a database really is: Predicates and propositions

3. Sistemas de gestión de bases de datos

Surge para evitar tener que rehacer los programas cada vez que cambiaba la base de datos. Se hizo evidente que la relación directa entre los programas y los datos es una gran desventaja. Por lo que aparece el gestor como punto intermedio

---

Definición: Es un conjunto de programas que gestiona y controla la creación, manipulación y acceso a la base de datos. Es una abstracción entre los programas y la base de datos.

Independencia de datos: Es la propiedad del SGBD consistente en que los cambios en la estructura de la base de datos no repercutan en los programas o sistemas de información que utilizan.

Funciones de los SGBDs

- Abstracción
- Almacenamiento y operaciones
  - ofrecer estructuras eficientes
  - ofrecer un lenguaje de consulta, una herramienta para manejarlo
- Seguridad: evitar accesos no autorizados
- Integridad: asegurar que la base queda integra (asegurar los datos a través de restricciones). *Registrar un director al agregar una película que nunca existió*
- Consistencia: accesos concurrentes por múltiples usuarios
- Recuperación: ofrecer herramientas para la recuperación ante fallas, usan un archivo de log, donde todo lo que hace el usuario queda registrado, de forma tal de que si pasa algo, se pueda recuperar
- soporte transaccional: que permita trabajar con transacciones, se realiza por completo la acción o no.

4. Arquitectura de 3 capas ANSI/SPARC

El ANSI/SPARC propuso una arquitectura de 3 niveles de abstracción para la descripción de una base de datos. Esta representación asegura la independencia.

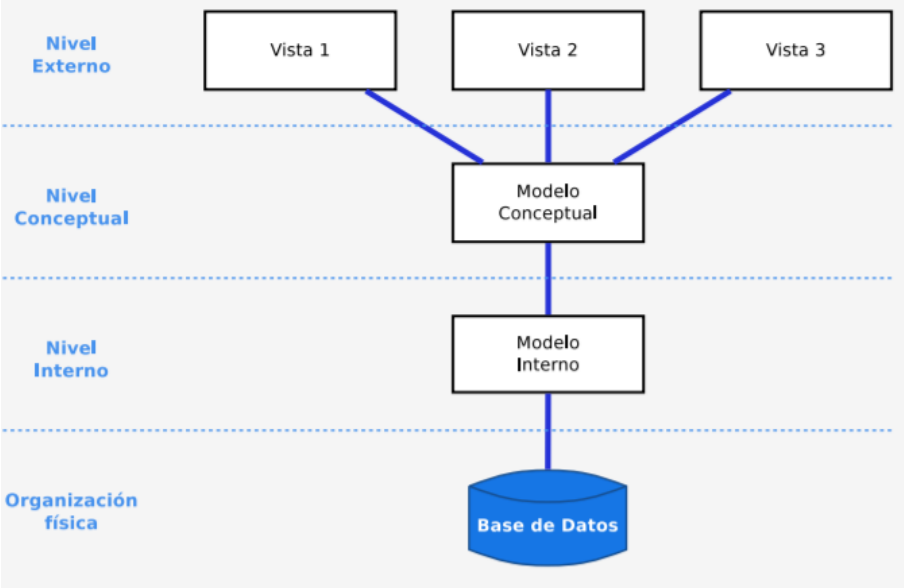


Figura 1: Modelo de 3 capas

Modelo interno

Representa la forma en que los datos se almacenan usando estructuras de datos.

Modelo conceptual

Describe la semántica de los datos abstrayéndose de su implementación física. Entidades, tipos de datos, operaciones restricciones de seguridad. Dice que tipo de cosas represento

Modelo externo

Representa la forma en que los usuarios perciben los datos

Modelo conceptual

Describe la semántica de los datos incluyendo sus características.
Un modelo de datos debe incluir:

- Conjunto de objetos, que tienen sus propiedades (atributos) y interrelaciones entre ellos que representa la estructura
- Un conjunto de operaciones para manipular los datos
- Restricciones sobre los objetos, las interrelaciones y las operaciones

Modelo Entidad-Interrelación

Esto es un modelo gráfico (Usamos la notación original del modelo de Chen)(diagrama de entidad-interrelación), es una herramienta comunicativa.

- Tipo de entidad: es un tipo de clase de objeto en particular. Los tipos de entidad en el diagrama los recuadramos. *Algunos ejemplos: futbolista, país, etc* El recuadro ya indica que tenemos esa entidad en la base de datos. No confundir con las instancias de entidad. *Ej Lionel Messi, Argentina*
- Atributo: Es una propiedad que describe a la entidad. Se representan con un circulo y una flecha de la entidad al atributo. *Ej. Futbolista → cotización, edad (fecha de nacimiento mejor!), nombre, club, ¿hijos? pensar quien esta pidiendo la base de datos, Asignatura → código, nombre, créditos, etc* Los atributos de una entidad tienen que surgir de lo que el cliente nos esta planteando.
- Tipo de interrelación: es la definición de un conjunto de relaciones o asociaciones similares entre dos o mas tipos de entidades. Rombo en el medio de la flecha, que indica que tipo es. *Ej.Futbolista - Nacio en - Pais.* No tenemos algo que indica en que orden leer el verbo.

La 'flecha' de una entidad al atributo que viene a ser otra entidad indica que hay un vinculo.
Cada entidad tendrá valores particulares para cada uno de los atributos. En el diagrama no indicamos que de que tipo son los atributos, lo ponemos en el diccionario.
El conjunto de valores lo podemos ver como una tupla, que surge del producto cartesiano de los dominios de los valores.

Valores Nulos

- Se puede indicar en una tabla si acepta valores nulos o desconocidos
- Un valor nula significa que se desconoce un valor de una fila, por ejemplo no se solicito un dato, o no se cargo migrando de otra tabla. A veces un valor nulo se utiliza para indicar que un dato no tiene valor (cosa distinta a tener valor y desconocerlo). *Ej. Lo podemos usar para indicar que algo esta vigente, que viene de otro país, etc*
- Al tener valores nulos, SQL trabaja con una lógica de tres valores, Verdadero, Falso, Nulo/Desconocido.

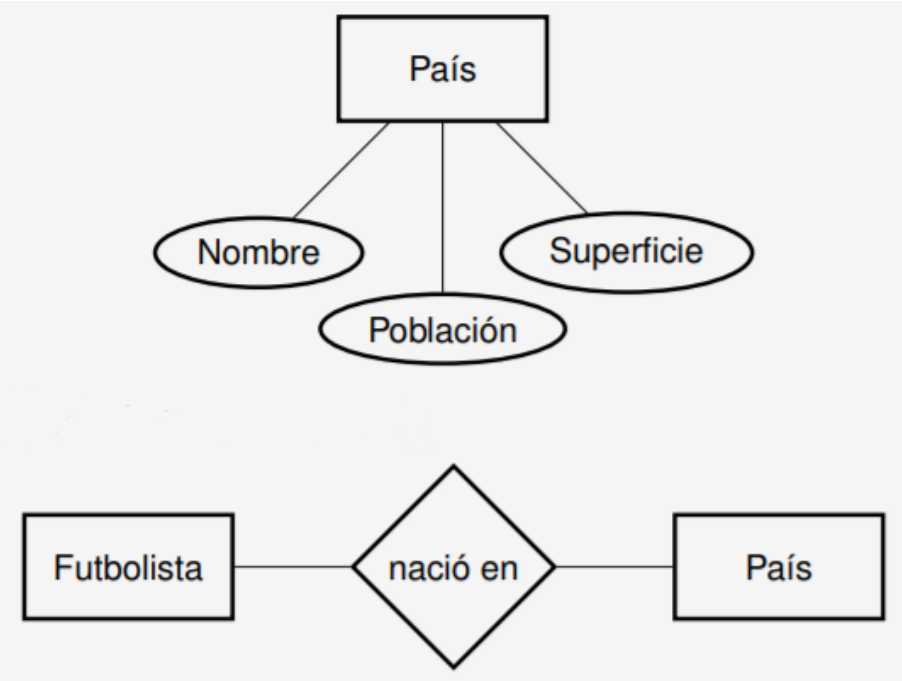


Figura 2: Ejemplos de diagramas

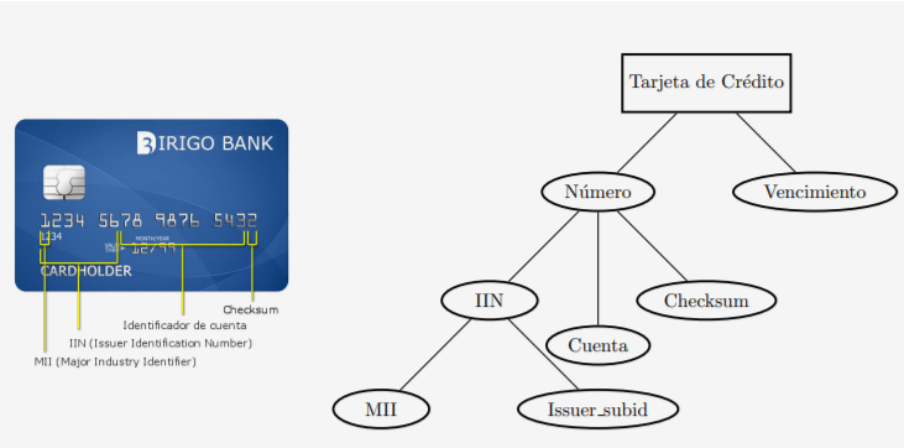
- Hay que tener cuidado al realizar consultas, ya que los valores nulos se tratan como desconocidos, por lo que las consultas no los incluirían en los resultados. Al momento de evaluarlos devuelve Falso. Se puede arreglar con el operador IS [NULL | NOT NULL]
- Si una columna puede tener valores nulos, la consulta armarla con la lógica de tres valores (usar el operador IS)
- En subconsultas algunos problemas con ellos son mas difíciles de detectar. Los EXISTS e IN no son tan intercambiables, por lo que hay que estar atento en su uso
- Cuando se intenta ordenar por una columna que tiene valores nulos, por defecto, los valores nulos se muestran al final. Si queremos cambiar el comportamiento, se agrega NULLS FIRST. Algo parecido ocurre cuando ordenamos descendentemente con DESC, se le puede agregar NULLS LAST
- En las funciones de agrupamiento y agregación los nulos por lo general se ignoran. No se suman ni incluyen en el promedio, no se devuelven ni como máximo no como mínimo.
- Si queremos que el COUNT los tenga en cuenta, hay que revisar que expresión se esta contando. (El COUNT cuenta filas)
- No es una instancia del dominio

Atributos compuestos

Nos puede interesar a los atributos en atributos mas simples. *Los números de tarjetas de créditos, cada uno representa algo.* Esto lo podemos hacer si es relevante para nuestro negocio.

Atributos multivaluados vs monovaluados

Teléfono y mail de contacto Podemos tener mas de una entidad de estos, se indican con doble borde del circulo de entidad.



Almacenados vs derivados

La densidad de población, si tenemos la población y la superficie ya la podemos obtener. Este se indica con un borde del circulo punteado.

Conjuntos de entidades

Es el conjunto de instancias de un determinado tipo de entidad en un estado determinado de la base de datos. *Por ejemplo con País, la base de datos puede tener cargada los siguientes países: Argentina, Italia, Países Bajos*

Restricciones

Queremos que toda entidad adentro del diagrama, tenga al menos un valor que siempre van a ser distintos entre si. Lo llamamos atributos claves o identificadores únicos. Si no lo encontramos, debemos crear uno. Estos permiten identificar unívocamente a las entidades. Los identificamos subrayándolos.

Hay una cuestión de minimalidad de la clave. El conjunto tiene que ser minimal, no debe tener ningún **subconjunto** del mismo que sea capaz de identificar unívocamente a los entidades. Aun así, pueden existir mas de un conjunto de atributos clave para un tipo de entidad. No tiene nada que ver con la longitud.

La propiedad de ser clave no depende del estado actual de la base de datos. Tiene que ser global, respecto de todos los estados posibles de la misma! En los exámenes por ahí aparece como clave el nombre, es con fines didácticos solamente.

Claves sustitutas

Claves sustitutas, artificiales, o "Surrogate Key", no es ni mala ni buena practica. El problema que tiene es que esta clave no existe en el mundo real, por lo que no sabemos si cargamos alguna fila dos veces. Priorizar usar los atributos existentes antes.

Es identificador único para cada registro de la tabla, que no surge de los datos, si no que es independiente de ellos y generado por el sistema. Por lo tanto, es una columna extra que se agrega a la tabla. Se diferencia de las claves naturales, porque las naturales tienen un significado contextual de lo que se busca representar. *Ej. Numero de padrón de alumno, código de materia son claves naturales debido a que tienen un significado contextual.* Las sustitutas, por lo tanto, no tienen un significado.

Cualquier valor generado por sistema que no permita duplicados en distintas filas, califica para serlo. Los casos mas comunes son un numero secuencial, o un numero aleatorio

Una clave sustituta se puede ver en el siguiente ejemplo con la columna 'Id', empieza en 1 y van aumentando de a 1, esta es una decisión arbitraria

Alumnos(Id,Legajo,Nombre)

Id	Legajo	Nombre
1	51253	Juan
2	51254	Lucas

Convenciones

- Clave primaria sustituta → Id
- Cada columna que haga referencia a otra columna de otra tabla → Id_*Nombre de la tabla referenciada en singular*

Recomendaciones

- No perder restricciones sobre la clave natural
- Definir la clave natural *El legajo* como clave candidata, para que no haya valores duplicados. Se puede hacer con un constraint de tipo UNIQUE
- Agregar restricción de que el legajo no sea nulo, con el constraint NOT NULL
- Puede ser conveniente definir índice con el legajo para facilitar búsquedas

¿Por que usarlas?

- Es decisión de diseño de la base de datos.

Desventajas

- Mayor uso de espacio debido a la nueva columna
- Afecta la performance de consultas, ya que entran menos registros en bloque de disco y porque algunas consultas deben empezar a usar mas *joins. Ej. Si un alumno quiere buscar sus notas, con el esquema de claves naturales, consulta la tabla de notas con su padrón, mientras que con las sustitutas hay que realizar un join con la tabla de alumnos ya que el vinculo es por id y no por padrón*
- En un esquema de claves sustitutas tenemos mas restricciones o constraints a validar, afectando la performance de actualizaciones
- Perdida de significado de claves foráneas

Ventajas

- Inmutabilidad ante cambios en los datos. *No importaría si cambia el formato del legajo, si cambia el legajo, hay que modificar todas las tablas*
- La performance de la base de datos (depende del caso particular), *Si tenemos muchas claves foráneas que usen muchos atributos o ocupen mucho, usar claves sustitutivas de un tipo de dato pequeño - un int - puede implicar una mejora de espacio que compense el agregado de la columna*

¿Que tipo de claves sustitutas usar?

Secuenciales

- Depende del sistema gestor de la base de datos, hay que revisar cual recomiendan. El incremento es automático
- La desventaja es con migraciones en bases de datos, puede ocurrir que algunos ids ya existan en el otro ambiente. Otro problema puede ocurrir con la seguridad, el tener acceso a los valores de una fila, puede otorgar información extra. También otro problema de seguridad, se da en poder crear fácilmente ids de registros fácilmente.

Claves aleatorias

- Para solucionar los problemas se pueden usar claves aleatorias. Se utiliza un rango grande para evitar colisiones. Se utiliza UUIDs (128 bits, rango muy grande)
- La desventaja es el espacio ocupado

Interrelaciones

- Cardinalidad: Con cuantas instancias de cada tipo de entidad pueden relacionarse con una instancia concreta de tipos de entidades restantes. El **máximo** numero de instancias posibles. *Un futbolista solo puede haber nacido en un único país. En un país pueden haber nacido muchos futbolistas*
- Participación: **Mínima** cantidad de instancias de cada tipo de entidad que deben relacionarse con una instancia concreta de los tipos de entidades restantes. Relación muy fuerte. *Un futbolista debe haber nacido en algún país. En un país puede no haber nacido ningún futbolista.*

Tener en cuenta que pueden existir tipos de interrelación recursivos o unarios.

Restricciones de cardinalidad + Restricciones de participación = Restricciones estructurales

Atributos (en interrelaciones)

Es un atributo que sale de la relación. (del rombo que esta en el medio indicando el tipo) *Fecha en Alumno aprobó Asignatura*

Atributos clave (en interrelaciones)

Si tomo dos instancias distintas, el valor del atributo debe de ser distinto. Son la forma de identificar los arcos (aristas) entre cada relación de instancias. *Ejemplo: Con la asignatura y el numero de padrón ya puedo identificar a todos los alumnos si aprobaron la materia.*

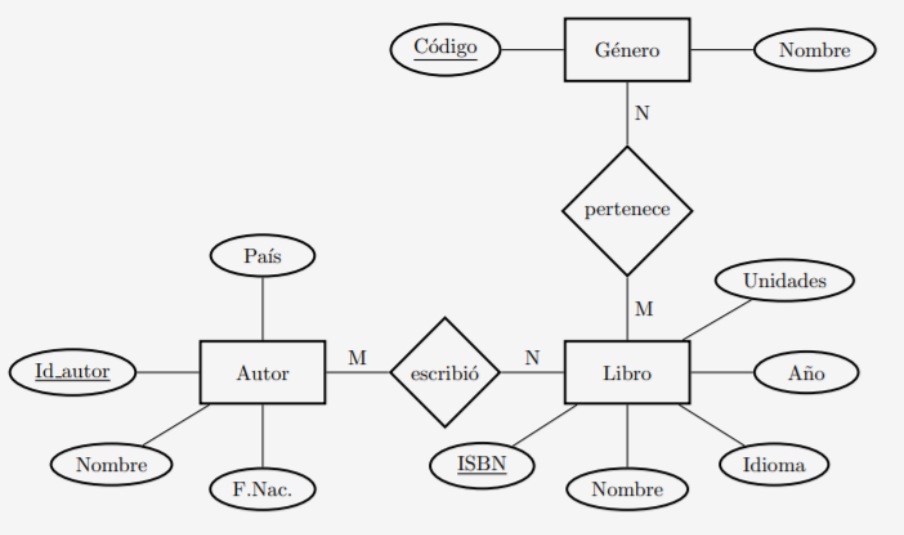
Sólo pueden formar parte de los atributos clave de una interrelación los atributos clave de los tipos de entidad que participan de la misma.

Pasos para resolver problemas

1. Identificar tipos de entidad
2. Identificar atributos
3. Identificar tipos de interrelación
4. Identificar atributos clave

5. Identificar restricciones estructurales

Ejemplo: Los dueños de esta librería desean crear una base de datos que contenga información sobre los libros actualmente en venta, y que permita hacer búsquedas por nombre o país de origen del autor, género, idioma y año.

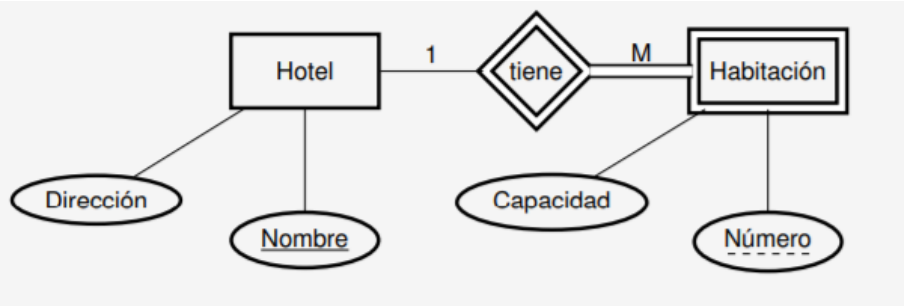


Modelo Entidad-Interrelación mas avanzado

Entidades fuertes y débiles

Las fuertes son aquellas que puedo identificar por si mismas. Las débiles dependen de otro para subsistir. Su clave se compone de la clave de su entidad identificadora, mas algún/os atributos propios, que se denominan discriminan tes y se indican con lineas punteadas. Un tipo de entidad débil tiene participación total en el tipo de interrelación que la vincula con su tipo de entidad identificadora.

Gráficamente a las entidades débiles se las indica con doble raya en el rectángulo, la flecha al rombo, y el rombo.



Interrelaciones ternarias

Son aquellas que participan 3 tipos de entidad distintos

La cardinalidad de un tipo entidad determina la cantidad de instancias de interrelación en que puede aparecer, fijadas las instancias de los otros dos tipos de entidades. Para identificarlos necesito los atributos clave de los tres tipos de entidad.

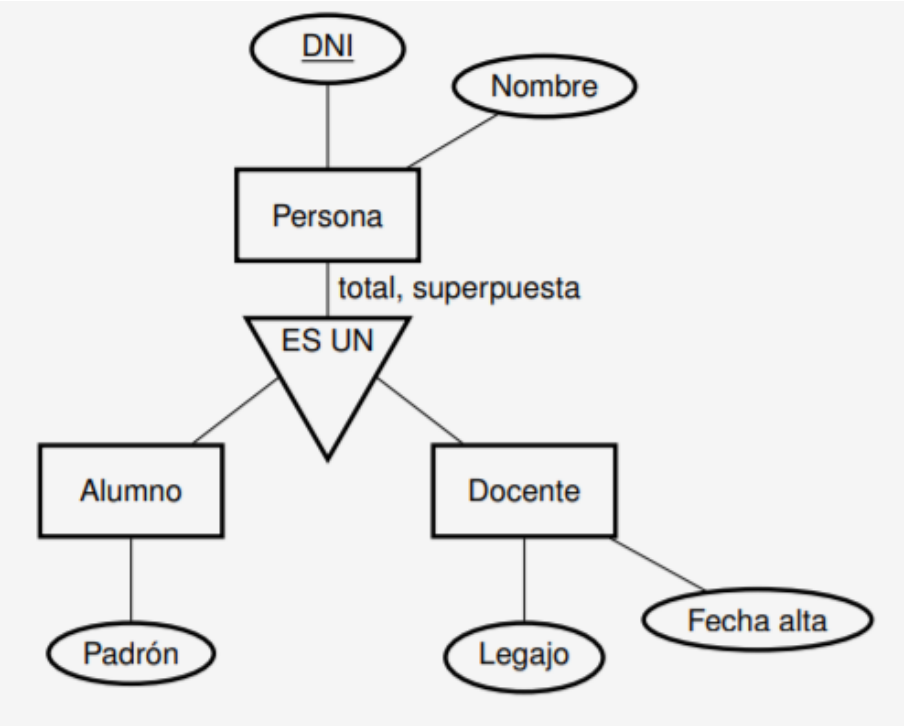
Agregación

La desventaja del modelo anterior es que *no nos permite registrar cantantes en rondas si no fueron calificados.* Se resuelve con una agregación. Se representa con una caja que agrupa a las entidades. Las entidades que se relacionen con esta, llevan la flecha hasta la caja. Se trata como que la *agregación de un Cantante y una Ronda fueran una entidad en sí misma.*

Generalización/especialización

Permiten evitar información redundante. Representan relaciones del tipo 'es un'. Gráficamente se muestran con un triangulo dado vuelta. Se heredan los atributos de la instancia padre. No hay herencia múltiple.

Persona es un alumno/docente, persona tiene como atributos al DNI y al nombre, mientras que el alumno y docente tienen sus atributos específicos.



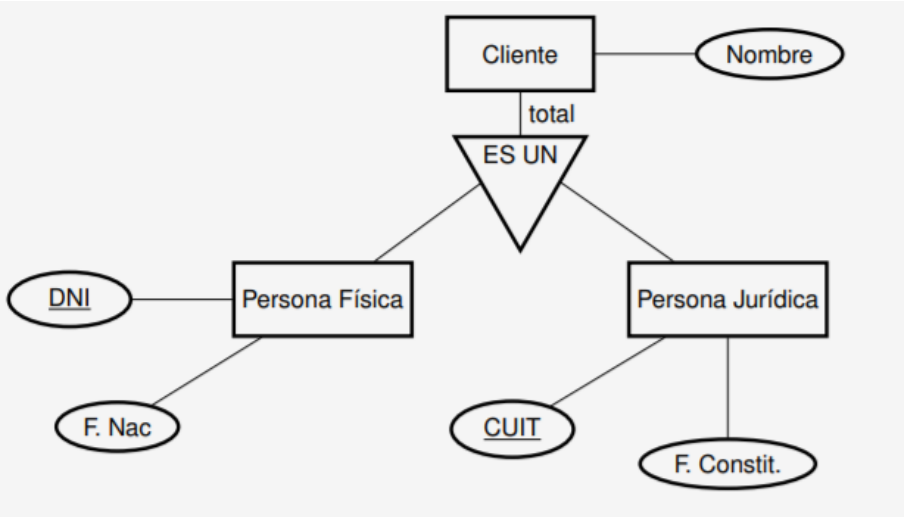
2 propiedades de este tipo de relación:

- Superposición: Los subtipos de entidad pueden ser disjuntos o superpuestos. En caso de ser superpuestos, una instancia del tipo de entidad padre puede corresponderse con instancias de varios subtipos de entidad.
- Completitud: Los subtipos de entidad pueden cubrir a todo el tipo de entidad padre (total), o no (parcial). En caso de no cubrirlo, puede ocurrir que algunas instancias del tipo de entidad padre no se correspondan con ningún subtipo de entidad.

Unión

En la unión también tenemos a un padre y distintos subtipos de entidad. La diferencia es que el padre es subclase de los subtipos de entidad (que son la superclase).

Un cliente ES UNA persona física o bien ES UNA persona jurídica. Pero NO necesariamente una persona física debe ser un cliente del banco.



Dependencia existencial

Se ve en el cardinal mínimo, necesito si o si de otra entidad para existir.

Clase 2

5. Practica: Ejercicio AFA y Taller I

Conceptos

- Cardinalidad: primer termino mínimo, segundo máximo en la cardinalidad (de ambos lados)
- Conjunto de entidad
- Entidad
- Atributos (claves, otros)
- Semántica, el significado de la interrelación
- Interrelación
- Conjunto de interrelación

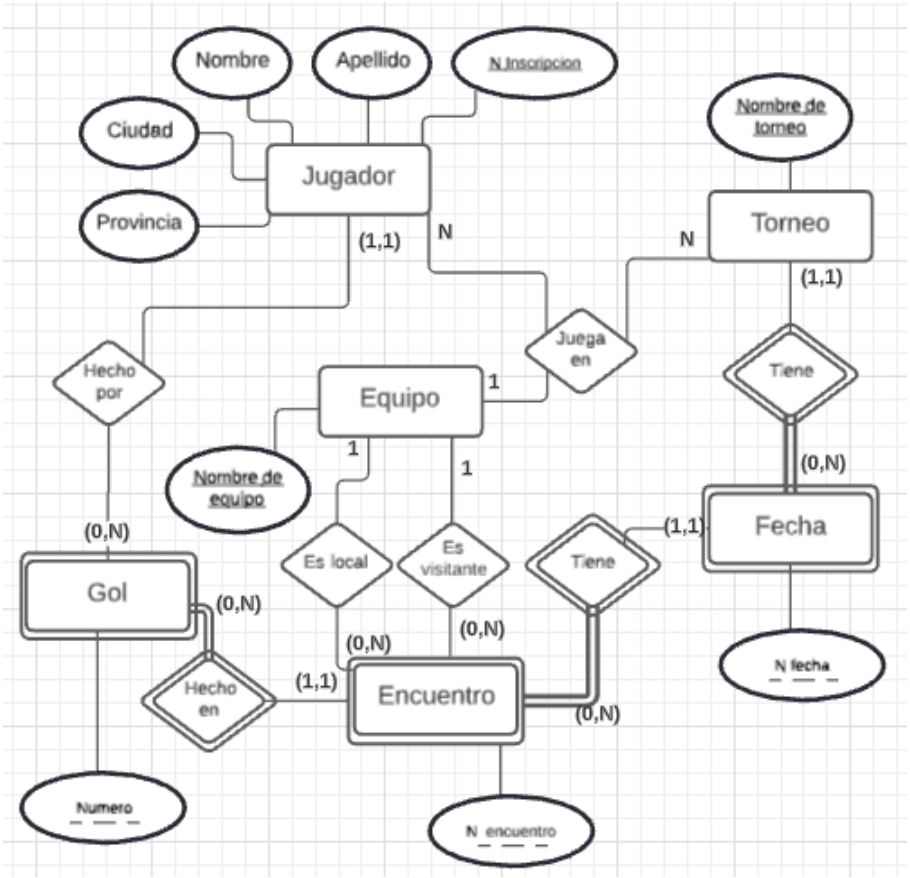
Notas

- La cardinalidad máxima tiene que estar si o si, se describe con esta siempre. Para enriquecer se agrega la cardinalidad mínima
- Una entidad débil puede ser a su vez una entidad fuerte de otra.
- No puedo tener un 0 como cardinalidad mínima en una entidad fuerte por la dependencia existencial.
- Prestar atención a los sustantivos y a los verbos. Para ver cuales son las entidades mas visibles, se puede contar cuantas veces aparece un sustantivo.
- En el diagrama los sustantivos los ponemos en singular, cuando lo pasamos a la tabla los ponemos en plural generalmente.

- En las relaciones ternarias usamos solo las cardinalidades máximas.
- Por mas que se tenga la misma cardinalidad como mínima y máxima, se ponen ambas (1,1)
- Siempre tienen que estar los identificadores únicos.
- Tratar de no usar abreviaturas.
- Lo que no entra en el diagrama, se anota aparte (en el diccionario de datos)

AFA

- Estamos modelando los torneos solamente. No irse de la narrativa de estos.
- Equipo y encuentro es una relación particular, una binaria que tratamos como ternaria.
- En este caso usamos 2 binarias para evitar esa relación particular



Taller

- Usamos schemas para ordenar las tablas lógicamente. (tiene otros motivos de seguridad y demás también)
- Las primeras lineas usarlas para documentación. Nombre del archivo, resumen del script, fecha creado y modificado, y autor. (medio innecesario los últimos tres si se usa git)

```
DROP TABLE IF EXISTS paradas -- Buena practica

CREATE TABLE paradas (
        cod_parada integer NOT NULL,
    longitud numeric NOT NULL,
    latitud numeric NOT NULL,
    tipo_parada varchar,
    calle varchar,
    altura integer,
    entre1 varchar,
    entre 2 varchar
);

ALTER TABLE paradas -- Se cambia el dueño de la tabla
        OWNER TO postgres;
```

- Al hacer consultas siempre tratar de poner un limit, en caso de tener tablas grandes. *Ejemplo:*

  ```
  SELECT ... limit 10;
  ```

- Al trabajar con *csv* tener cuidado con los '.' y ',' cuando tenemos numeros. Revisar tambien los simbolos que no sean estandar *por ejemplo la 'ñ'.*
- Para exportar los archivos, irían a un lugar publico. *En Windows: C:\Users\Public*

Articulo recomendado: Evolutionary Database Design - Martin Fowler

Clase 3

6. Modelo relacional

- Es un modelo a nivel lógico.
- Va a intentar captar que es un conjunto, que es una relación formada por las tuplas.
- Ahora la abstracción es un concepto matemático llamado relación. (de discreta)

- Dominio: Conjunto de valores homogéneos.
- Producto cartesiano: A x B, se define como el conjunto de pares (a,b) que cumplen que 'a' $\in$ A y 'b' $\in$ B

La cantidad de elementos que tiene es la cardinalidad de A por la de B. Llevándolo a código, al seleccionar Ciudades y Países en una consulta de SQL, se esta haciendo un producto cartesiano en el fondo, ya que se tienen todas las posibles combinaciones. Después se filtraran de acuerdo a la condición.

```
SELECT
FROM Ciudades, Paises
WHERE
```

Una relación es un subconjunto de un producto cartesiano. Aquellos elementos del producto cartesiano tales que la ciudad pertenece al país.

En el modelo relacional, un **nombre de relación** R junto con una **lista de atributos** asociados se denomina **esquema de relación** y se denota:

$$R(A\_1, A\_2, ..A\_n)$$

Películas(nombre,año,director, oscars)

- Con cada atributo tengo un dominio en particular asociado. *Ej. nombre película en string, año en los naturales positivos* Películas esta en el subconjunto del producto cartesiano de todos los atributos.
- Las **instancias de la relación** son los elementos tengo cargados
- Un elemento se llama una **tupla** si la tupla pertenece al subconjunto, es una instancia $t \in$ Películas
- El valor tomado por un atributo A en la tupla t se denota como t[A] o t.A *Por ejemplo*: t[director] = Quentin Tarantino (referencia al valor en la tupla t)
- La cardinalidad en una relación R, es la cantidad de tuplas que posee.
- Las relaciones las vemos como tabla, donde las columnas representan atributos y las filas representan tuplas. También se puede decir archivos, registros para filas, y campos para las columnas.

Restricciones

De dominio

- Las restricciones de dominio especifican que dado un atributo A de una relación R, el valor del atributo en una tupla t debe pertenecer al dominio dom(A). Dominios posibles son naturales, reales, caracteres, string, bool, fechas, etc
- Se puede permitir que los atributos tengan el valor nulo (NULL)
- Deben ser atómicos (los atributos), no hay atributos compuestos (tarjeta de crédito) o multivaluados (conjunto de valores).
- No pueden existir dos tuplas distintas que coincidan en los valores de todos sus atributos. Una tupla no puede estar dos veces.

Superclave

- Generalmente existe un subconjunto SK del conjunto de atributos de R que cumple la condición de que dadas dos tuplas s, t ∈ R, las mismas difieren en al menos uno de los atributos de SK. Si un subconjunto SK cumple esto, lo llamamos **superclave** de R.
- Nos interesan las superclaves que son minimales (no admiten ningún subconjunto propio con la misma propiedad - no podemos achicar mas). Las llamamos **claves candidatas o claves**. Se elige una como **clave primaria**
- Las claves primarias las subrayamos

Por ejemplo:

PRIMARY KEY (nombre_pelicula, nombre_director)

No es minimal ya que podríamos tener *por ejemplo Quentin Tarantino, Tarantino.*

De integridad

Un esquema de base de datos relacional S es un conjunto de esquemas de relación S junto con una serie de restricciones de integridad.

- **Restricciones de integridad de entidad:** la clave primaria de una relación no puede tomar el valor nulo. Quiere decir que tienen que ser identificables.
- **Restricción de integridad referencial:** aquello a lo que referencia tiene que existir en la otra tupla de S referenciada. *Cuando un conjunto de atributos FK de una relación R hace referencia a la clave primaria de otra relación S, entonces para toda tupla de R debe existir una tupla de S cuya clave primaria sea igual al valor de FK, a menos que todos los atributos de FK sean nulos.*

Por ejemplo: Puedo tener una película sin actuación, pero no puedo tener una actuación sin película.

- La clave foránea puede ser null.
- Para agregar claves foraneas al crear una tabla: *clave tipo REFERENCES tabla(atributo)*
- Al intentar borrar de la tabla original, no te permite ya que esta referenciada desde la otra tabla.
- Las claves foráneas las indicamos también con un subrayado punteado (de la cátedra)

Operaciones

Las operaciones del modelo relacional se especifican a través de lenguajes como el álgebra relacional o el cálculo relacional. (Sección 8) Podemos realizar consultas o actualizaciones (inserción, eliminación, modificación

- Las de consulta no violan ninguna restricción ya que no modifican ninguna relación.
- Las de inserción de tuplas pueden violar restricciones de dominio, unicidad, de integridad de entidad o referencial.
- Las de eliminación pueden violar la integridad referencial. Se puede rechazar la eliminación, eliminar en cascada, o poner NULL.
- En las de modificación:
  - Si se modifica una clave foránea, se debe verificar que sus nuevos valores referencien a una tupla existente de la relación referenciada, o bien sean todos nulos. De lo contrario se debería rechazar la operación.
  - Si se modifica una clave primaria, puede violarse cualquiera de las restricciones de integridad, y se combinan las situaciones indicadas para inserción y eliminación.

Estas restricciones llevan a lo que se conoce como **transacciones**. Son un conjunto de operaciones que quiero que el gestor me asegure que o bien se ejecuten, o no se hace nada. Si una transacción no puede terminar de realizarse porque una de sus operaciones viola alguna restricción de integridad, entonces debe dejarse la base de datos en el estado anterior al inicio de la misma.

Ejemplos

- Los atributos multivaluados los tengo que poner en una tabla distinta. (Puedo tener varios teléfonos, los pongo en una tabla distinta)
- En un atributo compuesto tengo que poner todos los atributos
- ejemplo gerente-departamento, clave candidata, no puede haber dos filas con el nombre del gerente, se le agrega la palabra UNIQUE
- ejemplo docentes relación ternaria

Clase 4

7. Practica: Modelo relacional

Vamos a ver como mapear de un modelo de conceptual de Chen al relacional. Obtener criterios para decidir cual es conveniente entre dos o mas modelos posibles.

- Las entidades fuertes del conceptual pasan a ser entidades del relacional.
- La primary key pasa al relacional subrayada también,
- Las relaciones binarias pasan también, tiene ambas ids como clave. Se subrayan cortado como foreign key (además del normal)
- Si una interrelación tiene un atributo, va adentro de la relación.
- La relación es un concepto matemático, la interrelación no (es mas conceptual).

Entidades A(idA,...) B(idB,...)

Relación R(idA,idB)

- En las relaciones unarios, para el id de la otra entidad (mismo tipo) se le pone otro nombre cambiado arbitrariamente.
- En el caso binario 1:N (donde el valor mínimo de una es 0), como primary key de la relación usamos el id de la relación que tiene (1,1). Si ponemos a los dos, no seria mínima (seria una super clave). Hay que analizar también si conviene mover el atributo de la relación en la que tiene (1,1).

Entidades A(idA,...) B(idB,...)

Relación R(idA,idB)

Pasa a estar como B(idB,...,idA)

- Esto es para minimizar la cantidad de tablas.
- Si tenemos el caso binario 1:N con opción de ninguno, la relación es necesaria, no se puede mover a una entidad. No esta bien ingresar algún carácter que indique que significa ninguno.

En el caso 1:1 hay varias opciones.

- No olvidarse de definir las claves candidatas dependiendo el caso.
- En las entidades débiles se tiene como PK el atributo discriminante y la clave de la entidad fuerte también (se marca como clave foránea además)

Relaciones ternarias

- En el caso de las relaciones ternarias N:N:N se ponen como clave las tres ids, y las tres son foreign key.
- En el caso 1:N:N se pone la relación con clave primaria de N y N, y la de 1 la tiene como foránea solo:

R(idA,idB,idC)

- En 1:1:N las relaciones tenemos:

R(idA,idB,idC) o R(idA,idB,idC)

- No podemos ahorrarnos la tabla ya que estaríamos forzando a que todos los elementos tengan una asociación
- En 1:1:1 las claves candidatas son:

R(idA,idB,idC) o R(idA,idB,idC) o R(idA,idB,idC)

- Todas son claves candidatas (las 3), la PK es solo una. Las claves candidatas son igual de importantes, quiero que todas se verifiquen. Elijo una como PK, pero las otras siguen estando (UNIQUE).
- En las especializaciones/generalizaciones (**caso disjunto y total (sin solapamiento)** - es uno o el otro), B es un A, y C es un A.
- Tenemos las tres entidades, B y C con sus atributos propios y no comunes.
- Los atributos comunes van en la tabla A.
- En A se puede agregar un atributo *tipo* para decir en que tabla voy a poder buscar el resto de la información. (como es total, esto se puede asegurar que el tipo no sea nulo)

Taller

- Si es una restricción de PK se le agrega el prefijo 'pk_' Se hace también para las foreign keys (fn_).
- Conviene definirlas como constraint, ya que es mas robusto que definirla en la creación inicial porque ante un error nos indica mas claramente el problema.

Ejemplo de agregado de una clave como CONSTRAINT (parte 2 y 4):

```
ALTER TABLE paradas ADD CONSTRAINT pk_paradas PRIMARY KEY(cod_parada);


ALTER TABLE colectivos_por_parada ADD CONSTRAINT fk_colectivos_por_parada
FOREIGN KEY(cod_parada) REFERENCES paradas(cod_parada);
```

Intente provocar una violación a la restricción de integridad de entidad de la tabla colectivos por parada a través de un INSERT.

```
INSERT INTO colectivos_por_parada VALUES(NULL,44)
```

Intente provocar una violación a la restricción de integridad referencial definida, a través de un INSERT en la tabla que considere apropiada.

```
INSERT INTO colectivos_por_parada VALUES(99999999,44);
```

Intente provocar una violación a la restricción de integridad referencial definida, a través de un DELETE en la tabla que considere apropiada

```
DELETE FROM paradas WHERE cod_parada=1000086;
```

Intente modificar el código de la parada 1004561 por el código 1007800 utilizando un script de UPDATE. ¿Es posible hacerlo? No, es referida por la tabla colectivos_por_parada todavia.

```
UPDATE paradas SET cod_parada=1007800 WHERE cod_parada=1004561;
```

Formas de delete (aplican tambien al ON UPDATE):

- ON DELETE NO ACTION: Si hay referencias a otra tabla, no la borra.
- ON DELETE CASCADE: Elimina las referencias también (las tuplas) en forma de cascada sucesivamente
- Esta también SET NULL (cuidado si tenemos NOT NULL, sigue valiendo) y SET DEFAULT

Las autoridades de la Dirección de Transito quieren que sea posible cambiar el código de una parada, modificando automáticamente todas las filas que hacen referencia a ella en otras tablas. Modifique para ello el script de CREATE TABLE, definiendo una CONSTRAINT de ON UPDATE en la tabla correspondiente.

```
ALTER TABLE colectivos_por_parada ADD CONSTRAINT fk_colectivos_por_parada
FOREIGN KEY(cod_parada) REFERENCES paradas(cod_parada)
ON UPDATE CASCADE;
```

Es posible actualizar la tabla ahora.

Clase 5

En realidad se dio durante la tercera clase debido a que caía un feriado en el día de la clase. (A partir de este punto no es del todo correcto el orden de clases mencionado en este apunte)

8. Álgebra relacional

- Es un lenguaje para interactuar con un modelo.
- Están los procedurales y los declarativos.
- El álgebra relacional es de tipo procedural, nos va a permitir hacer operaciones usando relaciones (los datos) en vez de números.
- Define operadores.
- Es la base de la optimización para ejecutar las consultas.
- SQL es un lenguaje de tipo declarativo, solamente le digo que quiero, el gestor se encarga de obtener el resultado.

Operaciones básicas

Una operación es una función cuyos operandos son relaciones y cuyo resultado es una relación.

Operador selección

- Es unario, deja de una relación solo aquellas filas que cumplan una condición. Se aplica a cada tupla de la relación.
- No reduce la cantidad de columnas, solo deja menos filas.
- Se usan condiciones atómicas.
- Se ponen como subíndice

PELÍCULAS

nombre_película	año	nombre_director	cant_oscars
Kill Bill	2003	Quentin Tarantino	0
Django Unchained	2012	Quentin Tarantino	2
Star Wars III	2005	George Lucas	0
Coco	2017	Lee Unkrich	2

$\downarrow \sigma_{cant\_oscars \geq 1}(\text{Películas})$

nombre_película	año	nombre_director	cant_oscars
Django Unchained	2012	Quentin Tarantino	2
Coco	2017	Lee Unkrich	2

Proyección

- Es unaria también.
- Dada una relación, devuelve una relación cuyas tuplas representan **los posibles valores** de los atributos de L indicados en R. *Remueve* tuplas duplicadas, no aparecen múltiples dos veces el mismo valor.

PELÍCULAS

nombre_película	año	nombre_director
Kill Bill	2003	Quentin Tarantino
Django Unchained	2012	Quentin Tarantino
Star Wars III	2005	George Lucas
Coco	2017	Lee Unkrich

$\pi_{nombre\_director}(Películas)$

nombre_director
Quentin Tarantino
George Lucas
Lee Unkrich

Asignación

- Un operador que nos va a ayudar a escribir consultas, rompe en pasos las diferentes operaciones al asignar el resultado a operaciones.

$$Directores\_Oscar \leftarrow \pi_{nombre\_director}(\sigma_{cant\_oscars>0}(Películas))$$

DIRECTORES_OSCAR

nombre_director
Quentin Tarantino
Lee Unkrich

Redenominacion

- Permite modificar los nombres de los atributos de una relación y/o nombre de la relación misma.
- Nos permite preparar el resultado para otra operación.

PELÍCULAS

nombre_película	cant_oscars
Kill Bill	0
Django Unchained	2
Star Wars III	0
Coco	2

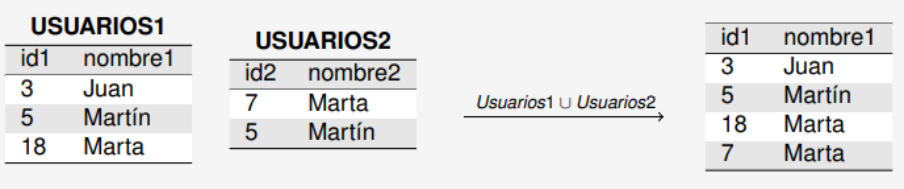
$\rho_{Films(film\_name,n\_oscars)}(Películas)$

FILMS

film_name	n_oscars
Kill Bill	0
Django Unchained	2
Star Wars III	0
Coco	2

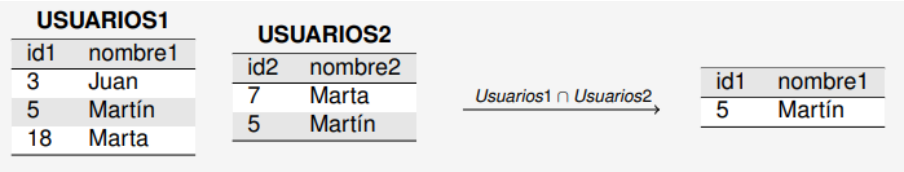
Unión

- Es la unión de dos conjuntos.
- Es necesario que tengan el mismo grado y coincidir en dominio (compatibilidad de union o tipo)



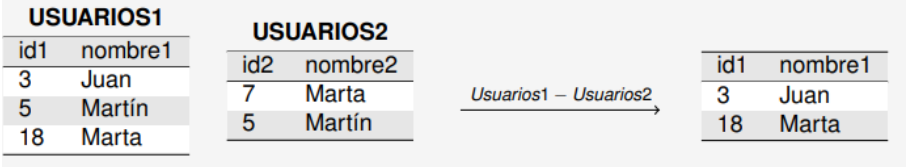
Intersección

- Devuelve aquellas tuplas que están en ambas.



Diferencia

- Conserva aquellas tuplas de R que no pertenecen a S. R - S



Producto cartesiano

- Son todas las posibles tuplas.
- Si hay atributos con el mismo nombre, tenemos que distinguirlo con el nombre de la relación.

PELÍCULAS

nombre_película	nombre_director
Kill Bill	Quentin Tarantino
Django Unchained	Quentin Tarantino
Star Wars III	George Lucas
Coco	Lee Unkrich

ACTUACIONES

nombre_película	nombre_actor
Kill Bill	Uma Thurman
Star Wars III	Natalie Portman

↓ Películas × Actuaciones

Películas.nombre_película	nombre_director	Actuaciones.nombre_película	nombre_actor
Kill Bill	Quentin Tarantino	Kill Bill	Uma Thurman
Kill Bill	Quentin Tarantino	Star Wars III	Natalie Portman
Django Unchained	Quentin Tarantino	Kill Bill	Uma Thurman
Django Unchained	Quentin Tarantino	Star Wars III	Natalie Portman
Star Wars III	George Lucas	Kill Bill	Uma Thurman
Star Wars III	George Lucas	Star Wars III	Natalie Portman
Coco	Lee Unkrich	Kill Bill	Uma Thurman
Coco	Lee Unkrich	Star Wars III	Natalie Portman

Junta

- Combina un producto cartesiano con una selección. Dadas dos relaciones y una condición, selecciona del producto cartesiano de las relaciones las tuplas que cumplen la condición.
- De condición de selección solo se admiten la conjunción de operaciones atómicas que incluye columnas de ambas relaciones. $=, \neq, \geq, \leq, <, >$ unidos por *and*.
- El caso mas general de operación de junta se denomina **junta theta (theta join)**.

PELÍCULAS

nombre_película	nombre_director
Kill Bill	Quentin Tarantino
Django Unchained	Quentin Tarantino
Star Wars III	George Lucas
Coco	Lee Unkrich

ACTUACIONES

nombre_película	nombre_actor
Kill Bill	Uma Thurman
Star Wars III	Natalie Portman

↓ (Películas $\bowtie_{nombre\_pelicula=nombre\_pelicula}$ Actuaciones)

Películas.nombre_película	nombre_director	Actuaciones.nombre_película	nombre_actor
Kill Bill	Quentin Tarantino	Kill Bill	Uma Thurman
Star Wars III	George Lucas	Star Wars III	Natalie Portman

Figura 3: Ejemplo de junta

Cuando la junta solo utiliza comparaciones de igualdad en sus condiciones atómicas, se denomina **junta por igual (equijoin)**. El resultado dispone de pares de atributos distintos que poseerán información redundante. Para liberarse de esto, se define la **junta natural** (evita que se repitan las columnas). No siempre se puede hacer, hay que asegurarse que los pares de atributos se llamen igual. Los atributos se llaman **atributos de junta**.

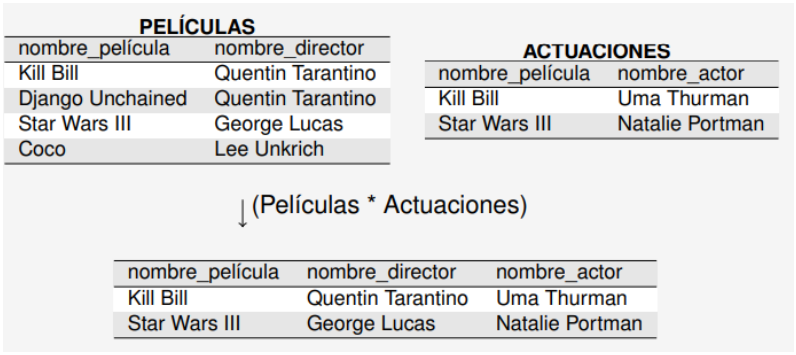
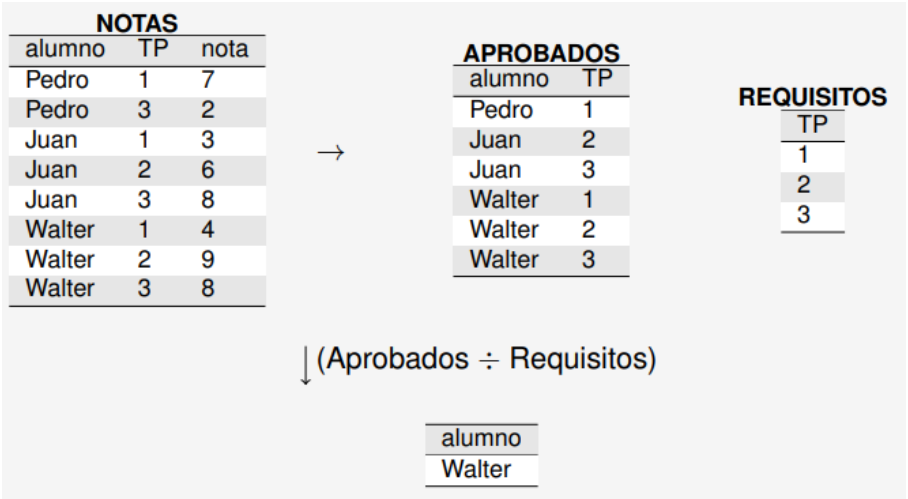


Figura 4: Ejemplo de junta natural

División (%)

- Busca encontrar aquellos elementos de un conjunto A que se vinculan con todos los elementos de un conjunto B. *Ej. A: Alumnos - B: Materias - Vinculo: Tiene nota*
- Lo que buscamos es que el producto cartesiano entre el resultado (tabla maximal), y el divisor, no se pase de la tabla original (dividendo) (incluido).
- Permite buscar quienes son todos los que cumplieron con una condición, o que no cumplieron ninguna.
- Tenemos dos relaciones, en la cual los atributos de la segunda están incluidos en la primera.



En SQL no se tiene el operador % de división, pero hay diferentes formas de encarar el problema.

- **Doble NOT EXISTS**: Surge de una equivalencia lógica. $\forall x : P(x) \leftrightarrow \neg\exists x : \neg P(x)$ El NOT EXISTS trabaja a nivel de filas, no le interesa que columnas devuelva.

```
SELECT padron, nombre, apellido FROM alumnos a
WHERE NOT EXISTS (
    SELECT 1 FROM materas m
    WHERE NOT EXISTS (
        SELECT 1 FROM notas n
```

```
        WHERE n.padron = a.padron
        AND n.codigo = m.codigo
        AND n.numero = m.numero
    )
)
```

- **Resta**: Surge de restar dos conjuntos para cada instancia. *En el ejemplo de los alumnos, para cada alumno, se resta todas las materias del sistema, y todas las materias en las que tiene nota el alumno. Si un alumno tiene nota en todas las materias, el conjunto es el vacio. Para revisar esto se usa NOT EXISTS*

```
SELECT padron, nombre, apellido FROM alumnos a
WHERE NOT EXISTS (
    (SELECT codigo, numero FROM materias)
    EXCEPT
    (SELECT codigo, numero FROM notas n WHERE n.padron = a.padron)
)
```

- **Con agrupación**:

Operaciones adicionales

Junta externa

La junta descarta a las tuplas de la relación que no se combinan con ninguna de las otras. La externa evita esto.

Tenemos tres variantes, la izquierda, la derecha, y la completa. Agregan valores nulos a los atributos que no se pudieron combinar. (izquierda solo a los de un lado, derecha a los del otro, completa a ambos.)

Tiene una condición para realizarlas.

9. Calculo Relacional

Es de tipo declarativo, nos da un lenguaje para describir que queremos obtener. Esta basado en la lógica de predicados.

Tiene diferentes partes:

- La lógica de predicados son funciones de varias variables que devuelven un valor de verdad.
- Tiene operaciones entre predicados, and, or, negado, entonces
- Tiene cuantificadores universales (para todo) para cualquier valor es verdadero, y existenciales (existe) si existe al menos un valor para el cual es verdadero.

De tuplas

Las variables representan tuplas. Un predicado simple es una función de una tupla o de atributos de tuplas, cuyo resultado es un valor de verdad. SQL se inspiro en esta forma de calculo.

Una expresión del calculo de tuplas tiene la siguiente forma:

$$\{t_1A_{11}, t_1A_{12}, \ldots, t_1A_{1k_1}, \ldots, t_nA_{nk_n} | p(t_1, \ldots, t_n, \ldots, t_{n+m})\}$$

Donde:

- p es un predicado valido
- $\{t_1, \ldots, t_n\}$ deben ser variables libres.
- $\{t_{n+1}, \ldots, t_{n+m}\}$ deben ser variables ligadas

Tiene llaves, y del lado derecho un predicado que toma variables. *{ / p($t_1, .., t_n, t_{n+1}, .., t_{n+m}$) }*. Tenemos las variables libres y ligadas, quiero que del lado izquierdo solo haya variables libres. (Lenguajes formales)

Ejemplos

- *{ t.name / NationalTeam(t) }*: Quiero los nombres que aparecen en NationalTeam. No me dice cual es el procedimiento para encontrarlo, solo decimos que queremos. (t es una variable libre, por lo que la puedo usar en el resultado)
- *{ p.name / Player(p) ∧ p.birth_ date<1980}*: p esta libre, donde m(p) = Player(p) ∧ p.birth_date<1980
- *{ t.name / Player(t) ∧ (∃ s) (Score(s) ∧ s.player_ id = t.id)}*: m(t) = (Score(s) ∧ s.player_id = t.id), s esta solo adentro de este m, esta ligada al cuantificador, por lo que no se puede utilizar del lado izquierdo.

Cuando cuantifico una variable no puede aparecer del lado izquierdo de la barra, se vuelve ligada.

Jugador mas anciano del mundial, las siguientes son equivalentes (es mas practico pensarlo con los existe):

- { p.name, p.birth_date | Player(p) ∧ (¬∃$p2$) (Player(p2) *and* p2.birth_date <p.birth_date) }
- { p.name, p.birth_date | Player(p) ∧ (∀$p2$) (¬Player(p2) *or* p2.birth_date ≥ p.birth_date) }
- { p.name, p.birth_date | Player(p) ∧ (∀$p2$) (Player(p2) → p2.birth_date ≥ p.birth_date) }

Llevándolo al álgebra relacional, podría verse así la selección:

- $\sigma(R)_{cond}$ = { t | R(t) ∧ cond}

Expresiones seguras

No toda expresión valida del calculo de tuplas es una expresión segura (safe expression). Por ejemplo:

$$\{p.name|\neg Player(p)\}$$

- Devuelve una cantidad infinita de tuplas como 'safsq' o 57.
- Una expresión segura es aquella que devuelve una cantidad finita de tuplas.

De Dominios

- Las variables representan atributos. Un predicado simple es una función de un conjunto de dominios cuyo resultado es V o F. Tiene los mismos cuantificadores y forma de expresión.
- Las variables que no quiero devolver las tengo que cuantificar.

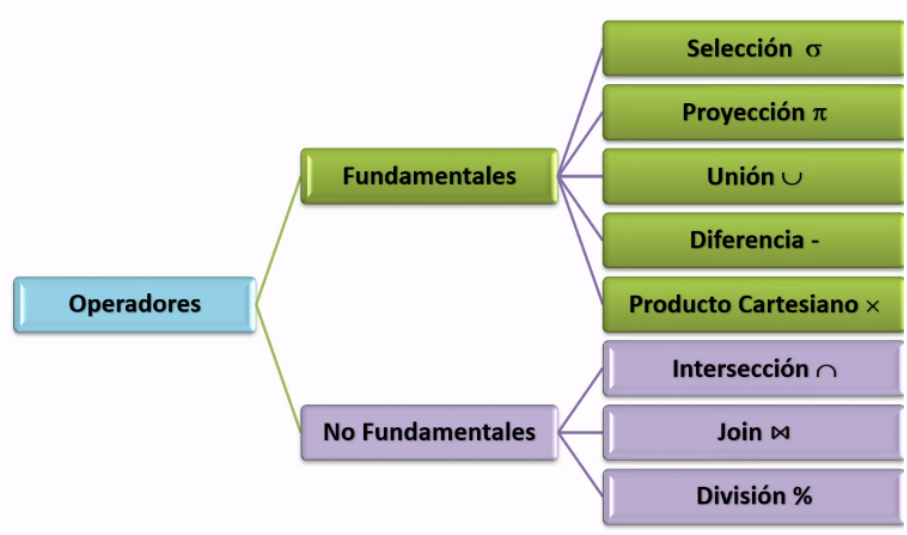
Completitud Relacional

Se demostró la equivalencia entre el álgebra relacional básica y el calculo relacional. Esto implica que ambos tienen el mismo poder expresivo.

Clase 6

10. Practica: Álgebra relacional

- El álgebra relacional es al SQL lo que el latín es a las lenguas romances. Es el fundamento.
- Proporciona un fundamento formal para las operaciones del modelo relacional el álgebra relacional.
- Dos tipos de operaciones, unarias y binarias.
- Unarias:selección, proyección
- Binarias: join y variantes, en conjuntos esta la unión, intersección, diferencia y producto cartesiano.
- Hay operadores fundamentales y no fundamentales.



Operadores en forma gráfica

División

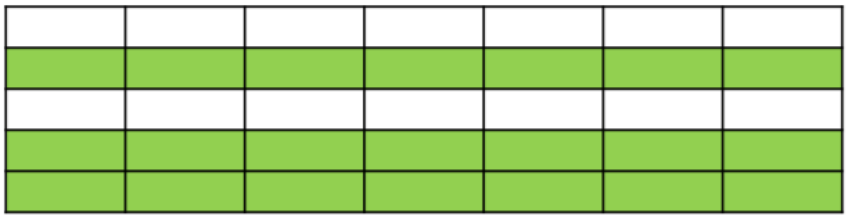
- Si surge la palabra *todos/todas* quiero relacionar dos conjuntos, surge la división.
- Es importante identificar de donde obtengo la información y no proyectar de menos/mas.

Forma general de la división

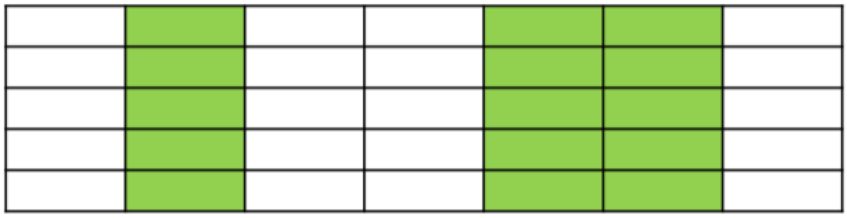
$$S \,\%\, T = \pi_{A1 \ldots A_n}(S) - \pi_{A1 \ldots A_n}(\pi_{A1 \ldots A_n}(S) \times T - S)$$

- No puedo dejar otra cosa que no quiero realmente.
- La proyección elimina los repetidos, no los muestra.

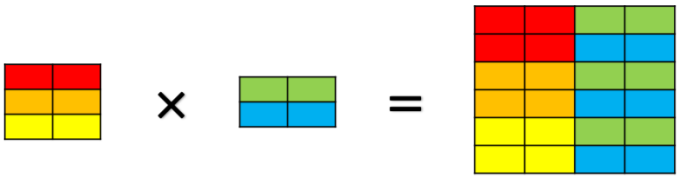
Selección σ



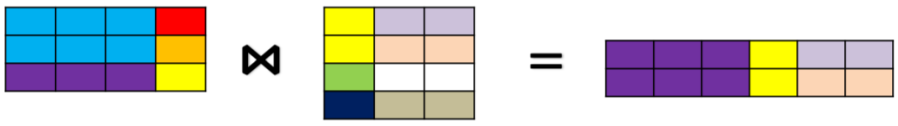
Proyección π

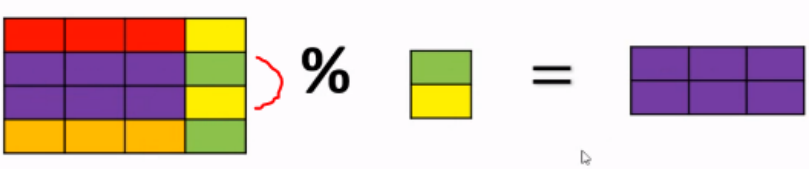


Producto Cartesiano ×



Join ⋈





Taller

Link para probarlo

- Listar las películas del año 2000.

σ year=2000 (movies)

- Mostrar el nombre y apellido de los directores de la base que tienen películas fechadas en el año 2000

peliculas2000 = σ year=2000 (movies)
idsPeliculas2000 = π id (peliculas2000)
relacionPeliConDirector = (idsPeliculas2000) $\bowtie$ id=movie_id (movies_directors)
idsDirectores2000 = π director_id (relacionPeliConDirector)
directores2000 = (idsDirectores2000) $\bowtie$ director_id=id (directors)
nombreApellidoDirectores2000 = π first_name,last_name (directores2000)
nombreApellidoDirectores2000

- Mostrar los nombres de las películas filmadas por Woody Allen

woodyAllen = σ first_name='Woody' and last_name='Allen' (directors)
idWoodyAllen = π id woodyAllen
relacionDirectorPeliculas = idWoodyAllen $\bowtie$ id = director_id (movies_directors)
idsPeliculasWoodyAllen = π movie_id (relacionDirectorPeliculas)
peliculasWoodyAllen = (idsPeliculasWoodyAllen) $\bowtie$ movie_id=id (movies)
nombrePeliculasWoodyAllen = π name (peliculasWoodyAllen)
nombrePeliculasWoodyAllen

- Mostrar los nombres de las películas en que Hitler figura como actor

actorHitler = σ last_name='Hitler' (actors)
idActorHitler = π id (actorHitler)
relacionActorRolesPeliculas = (idActorHitler) $\bowtie$ id=actor_id (roles)
idPeliculaActorHitler = π movie_id (relacionActorRolesPeliculas)
peliculasActorHitler = (idPeliculaActorHitler) $\bowtie$ movie_id=id (movies)
nombrePeliculasConHitler = π name (peliculasActorHitler)
nombrePeliculasConHitler

- Otra opción renombrando (hace un natural join)

ID_HITLER = π id (σ last_name = 'Hitler' actors)
ID_MOVIES = ρ id $\leftarrow$ movie_id (π movie_id (roles $\bowtie$ (ρ actor_id$\leftarrow$id ID_HITLER)))
π name (movies $\bowtie$ ID_MOVIES)

- ¿Algún director abarca todo los géneros?

directoresConGeneros = π director_id,genre (directors_genres)
generosPosibles = π genre (movies_genres)
directoresenTodosLosGeneros = directoresConGeneros $\div$ generosPosibles
directoresenTodosLosGeneros

- Mostrar el nombre y apellido de los directores que abarcaron (al menos) los mismos géneros que Polanski. ¿Y que Scorsese? ¿Y que Tarantino?

directorPolansky = σ last_name='Polanski' (directors)
idDirectorPolansky = π id (directorPolansky)
relacionGenerosPolansky = (idDirectorPolansky) $\bowtie$ id=director_id (directors_genres)
generosPolansky = π genre (relacionGenerosPolansky)
directoresConGeneros = π director_id,genre (directors_genres)
idDirectoresConMismosGeneros = (directoresConGeneros) $\div$ (generosPolansky)
directoresConMismosGeneros = (idDirectoresConMismosGeneros) $\bowtie$ director_id=id (directors)
nombresApellidosdirectores = π first_name,last_name (directoresConMismosGeneros)
σ last_name$\neq$'Polanski' (nombresApellidosdirectores)

- Mostrar el año de la ultima película.

aniosPeliculas = π year (movies)
aniosPeliculas2 = ρ year2$\leftarrow$movies.year aniosPeliculas
aniosDoble = aniosPeliculas $\times$ aniosPeliculas2
aniosMenores = σ year <year2 (aniosDoble)
ultimoAnio = π year aniosDoble - π year aniosMenores
ultimoAnio

- Listar las películas del ultimo año.

aniosPeliculas = π year (movies)
aniosPeliculas2 = ρ year2$\leftarrow$movies.year aniosPeliculas
aniosDoble = aniosPeliculas $\times$ aniosPeliculas2
aniosMenores = σ year <year2 (aniosDoble)
ultimoAnio = π year aniosDoble - π year aniosMenores
peliculasUltimoAnio = ultimoAnio $\bowtie$ movies
peliculasUltimoAnio

- Listar las películas del director Hitchcock en las que actuó Carroll.

directorHitchcock = σ last_name='Hitchcock' (directors)
idDirectorHitchcock = π id (directorHitchcock)
relacionPeliculasDirector = (idDirectorHitchcock) $\bowtie$ id=director_id (movies_directors)
idPeliculasHitchcock = π movie_id (relacionPeliculasDirector)
actoresEnPeliculasDeHitchcock = roles $\bowtie$ idPeliculasHitchcock
actorCarroll = σ first_name = 'Leo G.' and last_name='Carroll' (actors)
idActorCarroll = π id (actorCarroll)
rolesDeCarrollEnPeliculasDeHitcock = actoresEnPeliculasDeHitchcock $\bowtie$ roles.actor_id=actors.id idActorCarroll
idPeliculasBuscadas = π movie_id rolesDeCarrollEnPeliculasDeHitcock
peliculasBuscadas = movies $\bowtie$ id=movie_id idPeliculasBuscadas
peliculasBuscadas

- Listar las películas del director Hitchcock en las que NO actuó Carroll.

directorHitchcock = σ last_name='Hitchcock' (directors)
idDirectorHitchcock = π id (directorHitchcock)
relacionPeliculasDirector = (idDirectorHitchcock) $\bowtie$ id=director_id (movies_directors)
idPeliculasHitchcock = π movie_id (relacionPeliculasDirector)
actoresEnPeliculasDeHitchcock = roles $\bowtie$ idPeliculasHitchcock
idsActoresEnPeliculasDeHitchcock = π actor_id,movie_id actoresEnPeliculasDeHitchcock
actorCarroll = σ first_name = 'Leo G.' and last_name='Carroll' (actors)
idActorCarroll = π id (actorCarroll)

rolesDeCarrollEnPeliculasDeHitcock = actoresEnPeliculasDeHitchcock ⋈ roles.actor_id=actors.id idActorCarroll

idsRolesDeCarrollEnPeliculasDeHitcock = π actor_id,movie_id rolesDeCarrollEnPeliculasDeHitcock

peliculasSinCarrol = idsActoresEnPeliculasDeHitchcock - idsRolesDeCarrollEnPeliculasDeHitcock

idPeliculasBuscadas = π movie_id peliculasSinCarrol

peliculasBuscadas = movies ⋈ id=movie_id idPeliculasBuscadas

peliculasBuscadas

- Listar los actores que participan de al menos 3 películas.

La única forma de contar es realizando productos cartesianos.

PRIMER_PROD = π r.actor_id, r.movie_id, ro.movie_id (ρ r (roles) ⋈ r.actor_id = ro.actor_id and r.movie_id $\neq$ ro.movie_id ρ ro (roles))

SEGUNDO_PROD = PRIMER_PROD ⋈ r.actor_id = roles.actor_id and r.movie_id $\neq$ roles.movie_id and ro.movie_id $\neq$ roles.movie_id roles

IDS_ACTORES = ρ id←r.actor_id (π r.actor_id (SEGUNDO_PROD))

actors ⋈ IDS_ACTORES

Clase 7

11. SQL

- Es un lenguaje no procedural, basado en el calculo relacional de tuplas.
- Tiene 9 partes el ISO SQL actualmente, las mas importantes son 2, la de Foundation (2) y Information and Definition Schemas (11). Se las conoce como Core SQL.
- Es una gramática libre de contexto. Su sintaxis puede describirse a través de reglas de reglas de producción.
- Para estudiar esto lo mejor es ir a la documentación, tiene las opciones y ejemplos.

Tipos de dato (en el estándar)

- INTEGER
- SMALLINT
- FLAOT
- DOUBLE PRESICION
- NUMERIC(i,j): tipo numérico exacto. Permite especificar la precisión i y la escala j en dígitos.
- CHARACTER(n) longitud fija
- CHARACTER VARYING(n) longitud variable VARCHAR(n)
- DATE: yyyy-mm-dd
- TIME(i)
- TIMESTAMP(i)
- BOOLEAN: true, false, unknown - Logica de tres valores.
- CLOB: Character Large Object - Generalmente guardados aparte
- BLOB: Binary Large Object

El usuario puede definir tipos de dato también:

- CREATE DOMAIN CODIGO_PAIS AS CHAR(2)

Configuraciones

- Las columnas pueden configurarse con valores por defecto con DEFAULT o auto incrementales AUTO_INCREMENT.
- PRIMARY KEY
- UNIQUE: una columna o conjunto de columnas no puede estar con valores repetidos.
- En el modelo relacional una relación es un conjunto cuyos elementos son las tuplas. Por lo tanto, una tupla no puede estar repetida en una relación.
- En SQL esto no pasa, pueden repetirse. Se conoce como multiset o bag of tuples.

Consultas

- La consulta básica es SELECT ... FROM ... [WHERE ...]
- SELECT es equivalente a la proyección del álgebra relacional, y el WHERE a la selección del álgebra relacional. La diferencia es que el WHERE no elimina duplicados, no es estrictamente una proyección.

Condiciones en el WHERE

- Podemos comparar atributos con otros, o con valores.
- Realizar un pattern matching
- Que este o no en un conjunto
- Que este en rangos
- Que no sea nulo
- Que exista
- Que este o no en una tabla
- Permite poner otra consulta adentro también, podemos generar anidamientos de consultas.

Nota: el distinto se escribe con <>, y no se pone =NULL se pone es IS NULL

Otros

- Podemos usar alias para el FROM (tambien para el SELECT para redenominar atributos) con AS para las tablas.
- Se permiten operaciones adentro del SELECT (+,-,*,/, etc)
- Tenemos funciones de agregación: SUM, COUNT, AVG, MAX, MIN
- DISTINCT después del SELECT elimina duplicados
- Para comparar patrones LIKE

Join

- Clausula JOIN para evitar escribir todas las condiciones de junta. INNER JOIN ... ON, NATURAL JOIN, LEFT/RIGHT/FULL OUTER.
- Si no escribo OUTER, sql interpreta que queremos INNER.
- Se puede plantear con el WHERE tambien.

Operaciones de conjuntos

- UNION, INTERSECT, EXCEPT, deben de tener compatibilidad de tipos (misma cantidad de columnas). Si no se agrega ALL se eliminan duplicados

Ordenamiento y paginación

- ORDER BY por defecto es ascendente
- La forma estándar de la paginación es OFFSET n ROWS FETCH FIRST .. ROWS ONLY. Algunos SGBD implementan LIMIT o TOP

Agrupamiento

- Queremos hacer un resumen de ciertos datos.
- La agregación colapsa las tuplas que coinciden con una serie de atributos.
- Formato:

  ```
  SELECT ... FROM ... GROUP BY ... [HAVING ...]
  ```

- Puedo usar sin agregar los atributos por los que agrupo. El resto si los quiero los tengo que agregar. (Promedio, suma, cantidad, etc) (Resumir muchos valores en uno)
- HAVING es para filtrar con algún valor en particular, es una clausula opcional.
- Si en una agregación no se especifican atributos de agregación, el resultado tendrá una única tupla.

Subconsultas

```
SELECT ... FROM ...
WHERE A IN (SELECT X FROM ...); -- Debe devolver una única columna

SELECT ... FROM ...
WHERE A = (SELECT X FROM ...); -- Debe devolver sólo 1 fila!

SELECT ... FROM ...                    -- (feature opcional)
WHERE (A, B) IN (SELECT X, Y FROM ...); -- Debe devolver 2 columnas

SELECT ... FROM ...
WHERE A < [ SOME | ALL ] (SELECT X FROM ...);

SELECT ... FROM ...                          -- Devuelve FALSE/TRUE según
WHERE [ NOT ] EXISTS (SELECT ... FROM ...); -- la tabla esté vacía o no
```

- Dependencia con lo de afuera
- Si no depende se dice que no están correlacionadas, tiene menor costo.
- Me doy cuenta si esta correlacionada si usamos algo de la consulta mayor en la subconsulta. Puede pasar que en la ejecución el gestor la logre separar.

Tablas intermedias

Para poder usar el resultado de consultas intermedias, definimos esas tablas como sigue:

```
WITH nombre AS consulta
```

Esta tambien el WITH RECURSIVE que amplia el poder expresivo de SQL.

- No se guardan en ningún lado, son solo sintaxis. Seria lo mismo que poner la consulta adentro de la otra.
- Son mucho mas claras para documentar la consulta.

Inserciones

Es posible agregar valores a las tablar mediante:

```
INSERT INTO tabla VALUES (valores)
```

Se puede insertar también el resultado de un SELECT, los tipos tienen que ser compatibles

En cualquiera de los siguientes casos no se inserta el valor.

- Se asigna a una columna un valor fuera de su dominio
- Se omitió una columna que no podía ser NULL
- Se puso en NULL una columna que no podía ser NULL
- La clave primaria asignada ya existe en la tabla
- Una clave foránea hace referencia a una clave no existente

Eliminaciones

```
DELETE FROM ... FROM ...WHERE ...
```

Si se tenia ON DELETE RESTRICT no se elimina la fila.

Modificaciones

```
UPDATE ... SET ... WHERE ...
```

Para cada fila t que cumpla alguna condición de las siguientes no se actualiza.

- Se modifica una columna asignándole un valor fuera de su dominio
- Se pasó a NULL una columna que no podía tomar ese valor
- Se asignó a la clave primaria un valor que ya existe en la tabla
- Se modificó una clave foránea para hacer referencia a una clave no existente
- Se configuro ON UPDATE RESTRICT

Eliminando tablas

Para tablas: DROP TABLE tabla [RESTRICT|CASCADE]

Para esquemas: DROP SCHEMA esquema [RESTRICT|CASCADE]

Otras funciones

- SUBSTRING(string FROM start FOR length)
- UPPER/LOWER(string)
- CHAR_LENGTH(string)
- CAST(attr AS tipo)
- EXTRACT(campo FROM attr) - (para las fechas)
- || para concatenar

```
SELECT nro_factura ,
    CAST(
        CAST(año_venc AS CHAR) || '-' ||
        SUBSTRING(('0' || CAST(mes_venc AS VARCHAR)
                FROM (2 CAST(mes_venc <10 AS INTEGER)) FOR 2)
        || '-' ||
        SUBSTRING(('0' || CAST(dia_venc AS VARCHAR))
                FROM (2 CAST(dia_venc <10 AS INTEGER)) FOR 2) AS DATE
        ) AS fecha_venc
FROM Facturas f;
```

Estructura CASE

```
CASE WHEN .. THEN .. ELSE ..END
```

Permite agregar lógica de programación a la salida de una sentencia de sql.

Vistas en SQL

Podemos mostrar a los usuarios determinadas partes de la base de datos a través de las vistas. Se puede decir que es una tabla virtual, el resultado de una operación ejecutada en ese momento.

Se crean de la siguiente forma:

```
CREATE VIEW nombreVista
[(nuevoNombreColumna [,...])] AS consulta
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Ejemplo

```
CREATE VIEW EMP_ACC AS
SELECT Nombre, Documento, Sector, Sucursal FROM EMPLEADOS;
```

El usuario vería solo la información provista por:

```
SELECT * FROM EMP_ACC;
```

En las vistas también se pueden buscar casos específicos, agrupar, combinar, son condiciones que se agregan en el SELECT.

Se pueden eliminar vistas con

```
DROP VIEW nombreVista [RESTRICT | CASCADE]
```

RESTRICT si tengo algún objeto relacionado, no ejecuta el borrado CASCADE borra todos los objetos dependientes relacionados, aquellos que hagan referencia a la vista

Cambios en las vistas

Como los cambios en las tablas bases se ven automáticamente en las vistas, también se pueden modificar datos a través de las vistas y que modifique las tablas base. Para que pueda pasar esto, no debe de tener operadores conjuntistas, el operador DISTINCT, funciones agregadas, o GROUP BY

Para ser actualizable, FROM debe referenciar a solo una tabla base.

Materialización

Las vistas pueden tomar mucho tiempo si son complejas, se puede evitar con la materialización. Esto realiza una tabla temporal que almacena la vista. Se mantiene la vista por cada cambio de los datos.

¿Para que usarlas?

Las vistas se usarían para:

- Ocultar información
- Administración simple de permisos
- Personalizar datos
- Menor complejidad
- Proporcionar compatibilidad con las consultas ya armadas
- Integridad de los datos
- Combinar datos de servidores

Privilegios sobre ellas

Se otorgan privilegios a los usuarios de la siguiente forma

```
GRANT {Lista Privilegios | ALL PRIVILEGES}
ON NombreObjeto
TO {ListaIdentifiadoresAutorizacion | PUBLIC}
[WITH GRANT OPTION]
```

Se pueden revocar con:

```
REVOKE
```

Clase 8

12. Practica: Taller SQL

- Es un lenguaje declarativo, le digo que quiero y de alguna forma lo hace.
- ¿Cual forma de hacer algo es mas performante? No es fácil saberlo, generalmente el motor se puede dar cuenta cual hacer.
- Las cadenas de caracteres con comillas simples. Las comillas dobles las interpreta como que queremos acceder a una columna.
- Operadores lógicos AND, OR, NOT
- Evitar usar SELECT * ... , ya que no queda definida la consulta en ese momento.
- Definir forma de almacenar datos, esto es buena practica, pero trae problemas ante alguna equivocación.
- Con los operadores de conjuntos por defecto no hay duplicados.
- No usar NATURAL JOIN en aplicativos para evitar problemas a futuro, ya que la consulta no queda totalmente definida. Si el esquema de la tabla cambia, nos enteramos cuando se ejecuta.
- El resultado de una consulta se puede usar como tabla en la entrada de otra consulta (sub-consulta)
- Cuando agrupo puedo devolver alguna de las columnas agrupadas, o resultados que surgen de alguna función de agregación.

1 - Devuelva todos los datos de las notas que no sean de la materia 75.1

- Con que difiera alguno de ellos ya nos alcanza. Fijarse con De Morgan. Plantearlo con

  `AND`

  es un error común, ya que obligamos a que sea distinto si o si, en este caso hay que negar el resultado si queremos el

  `AND`

  .

```
SELECT * FROM notas WHERE codigo <> 75 OR numero <> 1
```

2 Devuelva para cada materia dos columnas: una llamada “código” que contenga una concatenación del código de departamento, un punto y el numero de materia, con el formato “XX.YY” (ambos valores con dos dígitos, agregando ceros a la izquierda en caso de ser necesario) y otra con el nombre de la materia.

```
SELECT to_char(codigo,'fm00') || '.' || to_char(numero,'fm00') AS "codigo",nombre
FROM materias
```

- Con el doble pipe lo estamos concatenando con un punto en el medio.
- Con el AS renombramos la columna inicial.
- Documentación del to_char.

3 - Para cada nota registrada, devuelva el padrón, código de departamento, número de materia, fecha y nota expresada como un valor entre 1 y 100.

```
SELECT padron,codigo,numero,fecha,nota*10 AS "nota"
FROM notas
```

4 - Ídem al anterior pero mostrando los resultados paginados en páginas de 5 resultados cada una, devolviendo la segunda página

```
SELECT padron,codigo,numero,fecha,nota*10 AS "nota"
FROM notas
OFFSET 5 ROWS FETCH FIRST 5 ROWS ONLY
```

- Esta

  ```
  LIMIT 5
  ```

  después del offset también.

- Si no le digo algún orden, no se que me va a devolver. Si queremos garantizarnos los resultados, hay que tomarse la costumbre de ordenar de la forma que no deje ambigüedad en las filas. En este caso agregando antes del

  ```
  FETCH
  ```

  un

  ```
  ORDER BY padron
  ```

  - Así evitamos tener un orden indeterminado.

```
SELECT padron,codigo,numero,fecha,nota*10 AS "nota"
FROM notas
ORDER BY padron,codigo,numero,fecha
OFFSET 5 ROWS FETCH FIRST 5 ROWS ONLY;
```

5 - Ejecute una consulta SQL que devuelva el padrón y nombre de los alumnos cuyo apellido es “Molina”

```
SELECT padron,nombre
FROM alumnos
WHERE apellido = 'Molina';
```

- No es lo mismo que: (el siguiente estaría mal)

```
SELECT padron,nombre
FROM alumnos
WHERE apellido = 'molina';
```

- Postgres es una base de datos CASE SENSITIVE. Por lo que no devolvería nada en este caso.

  ```
  LIKE
  ```

es case sensitive también, se puede usar

`ILIKE`

(no es estándar y no aprovecha índices)

```
SELECT padron,nombre
FROM alumnos
WHERE apellido ILIKE 'Molina';
```

- Otra alternativa es llevándolo a una forma común con Upper o Lower.

```
SELECT padron,nombre
FROM alumnos
WHERE Upper(apellido) = 'MOLINA';
```

6 - Obtener el padrón de los alumnos que ingresaron a la facultad en el año 2010

```
SELECT padron
FROM alumnos
WHERE fecha_ingreso>='2010-01-01' AND fecha_ingreso<='2010-12-31';
```

Con otra sintaxis:

```
SELECT padron
FROM alumnos
WHERE fecha_ingreso BETWEEN '2010-01-01' AND '2010-12-31';
```

7 - Obtener la mejor nota registrada en la materia 75.15

```
SELECT MAX(nota)
FROM notas
WHERE codigo=75 AND numero=15;
```

8 - Obtener el promedio de notas de las materias del departamento de código 75

```
SELECT AVG(nota)
FROM notas
WHERE codigo=75;
```

9 - Obtener el promedio de nota de aprobación de las materias del departamento de código 75.

```
SELECT AVG(nota)
FROM notas
WHERE codigo=75 and nota>=4;
```

10 - Obtener la cantidad de alumnos que tienen al menos una nota

```
SELECT COUNT(DISTINCT padron)
FROM notas
```

- COUNT

  solo cuenta los distintos de nulos, si queremos que sean distintas lo hacemos con

  DISTINCT

  .

11 - Devolver los padrones de los alumnos que no registran nota en materias

- Estamos realizando la operación de conjuntos: A - B

```
(SELECT padron
FROM alumnos)
EXCEPT
(SELECT padron
FROM notas)
```

```
SELECT padron
FROM alumnos
WHERE padron
NOT IN
(SELECT DISTINC(padron) FROM notas)
```

12 - Con el objetivo de traducir a otro idioma los nombres de materias y departamentos, devolver en una única consulta los nombres de todas las materias y de todos los departamentos

```
(SELECT nombre
FROM materias)
UNION
(SELECT nombre
FROM departamentos)
```

13 - Devolver para cada materia su nombre y el nombre del departamento

- Con

  JOIN

```
SELECT m.nombre, d.nombre
FROM materias AS m, departamentos AS d
WHERE m.codigo=d.codigo
```

- Otras forma para hacerlo:

```
SELECT m.nombre, d.nombre
FROM materias AS m INNER JOIN  departamentos AS d
ON m.codigo=d.codigo
```

```
SELECT m.nombre, d.nombre
FROM materias AS m INNER JOIN departamentos AS d
USING (codigo)
```

14 - Para cada 10 registrado, devuelva el padrón y nombre del alumno y el nombre de la materia correspondientes a dicha nota

```
SELECT a.padron,a.nombre,m.nombre
FROM alumnos AS a, notas AS n, materias AS m
WHERE n.nota=10 AND a.padron=n.padron
AND n.codigo=m.codigo AND n.numero=m.numero;
```

```
SELECT a.padron,a.nombre,m.nombre
FROM notas AS n
INNER JOIN materias AS m ON m.numero=n.numero AND m.codigo=m.codigo
INNER JOIN alumnos AS a ON a.padron=n.padron
WHERE n.nota=10;
```

```
SELECT n.padron,a.nombre AS nombre_alumno,m.nombre AS nombre_materia
FROM (notas n INNER JOIN alumnos a USING (padron)) INNER JOIN materias m
USING (codigo,numero)
WHERE n.nota = 10
```

15 - Listar para cada carrera su nombre y el padrón de los alumnos que estén anotados en ella. Incluir también las carreras sin alumnos inscriptos

```
SELECT c.nombre,i.padron
FROM carreras AS c LEFT OUTER JOIN inscripto_en AS i
USING (codigo)
```

```
SELECT carreras.nombre,alumnos.nombre,alumnos.apellido
FROM carreras
LEFT OUTER JOIN inscripto_en ON inscripto_en.codigo = carreras.codigo
LEFT OUTER JOIN alumnos ON inscripto_en.padron = alumnos.padron
```

16 - Ídem punto anterior pero teniendo en cuenta únicamente alumnos con padrón mayor a 75000

```
SELECT carreras.nombre,alumnos.nombre,alumnos.apellido
FROM carreras
LEFT OUTER JOIN inscripto_en ON inscripto_en.codigo = carreras.codigo
LEFT OUTER JOIN alumnos ON inscripto_en.padron = alumnos.padron
WHERE alumnos.padron > 75000
```

```sql
SELECT c.nombre,i.padron
FROM carreras AS c LEFT OUTER JOIN inscripto_en AS i
USING (codigo)
WHERE padron > 75000
```

```sql
SELECT c.nombre, ie.padron
FROM taller_4.CARRERAS c
LEFT OUTER JOIN taller_4.inscripto_en ie ON (ie.codigo = c.codigo AND ie.padron > 75000);
```

17 - Listar el padrón de aquellos alumnos que tengan m´as de una nota en la materia 75.15

- De esta forma no se tiene la mejor performance, ya que se esta revisando fila por fila con el valor de la consulta mayor.

```sql
SELECT a.padron
FROM alumnos as a
WHERE
(SELECT count(n.nota)>1 FROM notas AS n
WHERE n.padron=a.padron AND n.codigo=75 AND n.numero=15)
```

- Otra opción poniendo el >1 afuera de la subconsulta.

```sql
SELECT a.padron
FROM alumnos as a
WHERE
(SELECT count(n.nota) FROM notas AS n
WHERE n.padron=a.padron AND n.codigo=75 AND n.numero=15) >1
```

```sql
SELECT DISTINCT n1.padron
FROM notas n1, notas n2
WHERE n1.codigo = 75 AND n2.codigo = 75
AND n1.numero = 15 AND n2.numero = 15
AND n1.padron = n2.padron
AND n1.fecha <> n2.fecha
```

18 - Obtenga el padrón y nombre de los alumnos que aprobaron la materia 71.14 y no aprobaron la materia 75.15

```sql
SELECT A.PADRON, A.NOMBRE
FROM ALUMNOS A
WHERE EXISTS(SELECT 1 FROM NOTAS N WHERE N.NOTA >= 4
             AND N.CODIGO = 71 AND N.NUMERO = 14
             AND N.PADRON = A.PADRON)
AND NOT EXISTS(SELECT 1 FROM NOTAS N WHERE N.NOTA >= 4
             AND N.CODIGO = 71 AND N.NUMERO = 15
             AND N.PADRON = A.PADRON);
```

- Se fija que exista un aprobado en la materia, y que no exista un aprobado en la otra.

Otra forma:

```
SELECT A.PADRON, A.NOMBRE
FROM ALUMNOS A
WHERE A.PADRON IN (SELECT N.PADRON FROM NOTAS N WHERE N.NOTA >= 4
                          AND N.CODIGO = 71 AND N.NUMERO = 14 )
AND A.PADRON NOT IN (SELECT N.PADRON FROM NOTAS N
                                  WHERE N.NOTA >= 4
                                  AND N.CODIGO = 71 AND N.NUMERO = 15 );
```

19 - Obtener, sin repeticiones, todos los pares de padrones de alumnos tales que ambos alumnos rindieron la misma materia el mismo día. Devuelva también la fecha y el código y numero de la materia

```
SELECT DISTINCT N.PADRON PADRON, NOTA.PADRON PADRON2,
N.FECHA, N.CODIGO, N.NUMERO FROM TALLER_4.NOTAS N
INNER JOIN
TALLER_4.NOTAS AS NOTA
ON NOTA.PADRON < N.PADRON
AND NOTA.FECHA = N.FECHA
AND N.CODIGO = NOTA.CODIGO
AND N.NUMERO = NOTA.NUMERO;
```

20 - Para cada departamento, devuelva su código, nombre, la cantidad de materias que tiene y la cantidad total de notas registradas en materias del departamento. Ordene por la cantidad de materias descendente

```
SELECT d.codigo, d.nombre, COUNT(DISTINCT n.numero) AS "cant_mat",
COUNT(*) AS "cant_not"
FROM notas n INNER JOIN materias m USING (codigo, numero)
INNER JOIN departamentos d USING (codigo)
GROUP BY d.codigo, d.nombre
ORDER BY COUNT(DISTINCT n.numero) DESC
```

```
WITH cant AS (
        SELECT codigo, COUNT(numero) AS cant FROM materias GROUP BY codigo
), notasD AS (
        SELECT codigo, COUNT(notas) AS cant FROM notas GROUP BY codigo
)
SELECT d.codigo,d.nombre, c.cant, n.cant
FROM departamentos d, cant c, notasD n
WHERE d.codigo = c.codigo AND d.codigo=n.codigo
ORDER BY c.cant DESC
```

21 - Para cada carrera devuelva su nombre y la cantidad de alumnos inscriptos. Incluya las carreras sin alumnos

```
SELECT c.nombre, COUNT(i.padron)
FROM carreras c LEFT OUTER JOIN  inscripto_en i USING (codigo)
GROUP BY c.codigo, c.nombre
```

Es importante contar un atributo que no tenga nulos, así lo cuentan.

22 - Para cada alumno con al menos tres notas, devuelva su padrón, nombre, promedio de notas y mejor nota registrada

```
SELECT a.padron, a.nombre, AVG(n.nota) AS promedio, MAX(n.nota) AS mejor_nota
FROM alumnos AS a, notas AS n
WHERE a.padron = n.padron
GROUP BY a.padron, a.nombre
HAVING COUNT(*) >= 3
```

- Condiciones que se aplican grupo a grupo no van en el WHERE, ya que viene en una etapa después.
- Puedo evaluar cosas que después no devuelvo en el HAVING.

23 - Obtener el código y numero de la o las materias con mayor cantidad de notas registradas.

```
SELECT n.codigo, n.numero, COUNT(*)
FROM notas n
GROUP BY n.codigo, n.numero
HAVING COUNT(*) >= ALL (
    SELECT COUNT(*)
    FROM notas n
    GROUP BY n.codigo, n.numero
)
```

```
WITH cantidades AS (SELECT n.codigo, n.numero, COUNT(*) AS cant
        FROM notas n
        GROUP BY n.codigo, n.numero)
SELECT codigo, numero, cant
FROM cantidades c
WHERE c.cant = (SELECT MAX(cant) FROM cantidades)
```

- No es valido ordenarlo y devolver el primero, puede darse el caso de que varios coincidan con el mejor valor. Queremos todas con la mayor cantidad.
- No esta permitido realizar un MAX del COUNT en el HAVING.
- No podemos anidar funciones de agregación.

24 - Obtener el padrón de los alumnos que tienen nota en todas las materias

```
SELECT padron
FROM alumnos a
WHERE NOT EXISTS (
    SELECT *
    FROM materias m
    WHERE NOT EXISTS (
        SELECT *
        FROM notas n
        WHERE n.padron = a.padron
                    AND n.codigo = m.codigo
                    AND n.numero = m.numero
        )
)
```

- Es una división.
- Buscamos los alumnos con nota en todas las materias, tal que para los alumnos no exista materia en la que no tiene nota.
- Alumnos para los que no existe una materia en la que no existe una nota de ese alumno en esa materia

```
SELECT padron
FROM alumnos a
WHERE NOT EXISTS (
    (SELECT codigo, numero FROM materias)
    EXCEPT
    (SELECT codigo, numero FROM notas n WHERE n.padron=a.padron)
)
```

- Primer conjunto: todas las materias
- Segundo conjunto: todas las materias en las que un alumno tiene nota

```
SELECT padron
FROM notas n
GROUP BY n.padron
HAVING COUNT( DISTINCT(to_char(codigo,'fm00') ||'.'|| to_char(numero,'fm00') ))
       =
       (SELECT COUNT(*) FROM materias)
```

- Contar cuantas materias hay y cuento en cuantas materias tiene nota. Tiene que dar lo mismo
- El DISTINCT solo recibe un atributo, no es estandar que tome dos.

25 - Obtener el promedio general de notas por alumno (cuantas notas tiene en promedio un alumno), considerando únicamente alumnos con al menos una nota

```
SELECT CAST(COUNT(*) AS DECIMAL) / COUNT(DISTINCT padron)
FROM notas n;
```

Cuidado con realizar la división entera, esta trunca el resultado.

Clase 9

13. Practica: Taller Valores Nulos

- La política del estándar del NULL es ignorarlo en las consultas. Si son todos NULL, ahi da NULL.
- Código para crear las tablas e insertar los valores.

```
CREATE TABLE  clientes (
  cod_cliente integer             NOT NULL,
  nombre      character varying   NOT NULL,
  saldo       float8              NULL    ,
  localidad   character varying   NULL
)
```

```
INSERT INTO clientes(
        cod_cliente, nombre, saldo, localidad)
        VALUES (10, 'Juan' , 20    , NULL ),
               (20, 'Pablo', 40    , NULL ),
               (30, 'Ana'  , NULL  , 'Mar del Plata')
        ;
```

2) Las preguntas pasan a tener distinto significado con valores nulos, cambia la semántica.

1. El saldo total de los clientes es 60
2. Matemáticamente el saldo de Ana tiene que ser 0. Aun ignorando los valores nulos estos afectan en cierta forma.
3. Depende de la politica. Da 30, ignora los valores nulos. Para que sea correcto Ana deberia de tener saldo 30. (Llegamos a un absurdo, tiene 0 en uno, 30 en este)
4. Maximo 40, minimo 20. Estan condicionando el saldo de Ana dentro de estos limites.

3) En el orden hay 3 políticas, los deja al final, al comienzo, o lo deja donde lo encontró. Esto tiene implicaciones también.

4) La consulta nos da en realidad los clientes que **sabemos que viven** en Mar del Plata, no nos da todos.

```
SELECT *
FROM clientes
WHERE localidad ILIKE 'mar del plata';
```

5) El conjunto mas su complemento no me pueden dar el universo. Esto es por los valores nulos.

```
SELECT *
FROM clientes
WHERE localidad ILIKE 'mar del plata'
UNION
SELECT *
FROM clientes
WHERE localidad NOT ILIKE 'mar del plata';
```

- NULL esta en el rango? Da NULL, no lo toma.
- Hay que tener mucho cuidado al pasar de un diseño sin valores nulos a uno con nulos.

8) Una tautología puede dar falso debido a los valores nulos

```
--  E8 (Una Tautologías puede dar Falso!)
--  idea  adaptad de : C. J. Date-SQL and Relational Theory-O'Reilly (2012)
-- Ejecute la sentencia:
DELETE FROM taller_05.fabricas WHERE cod_fabrica = 40;
-- Luego obtenga todos los pares (cod cliente, cod fabrica) tales que
-- no se encuentren en la misma ciudad, o bien la fabrica no se encuentre
-- en Rosario. Analice el resultado e indique si el mismo se contrapone a la realidad.
-- veamos esta consulta:
SELECT cod_cliente, cod_fabrica
FROM taller_05.clientes c , taller_05.fabricas f
WHERE c.localidad <> f.localidad
   OR f.localidad <> 'ROSARIO'

-- estudiemos estos dos predicados:
--   (c.localidad <> f.localidad)  OR  (f.localidad <> 'ROSARIO')
-- cual es la respuesta?
-- pero si en el mundo real suponemos :
--  H0: la fabrica 50 esta en Rosarios
--  H1: la fabrica 50 no esta en Rosario
-- observar que  en el mundo real no existe otra hipotesis posible!
-- entonces:
-- si se cumple la la H0: el primer predicado es Verdadero y el segundo predicado es Falso
-- y me da ANA (<30 , 50>)
-- si se cumple la H1: el primer predicado es Falso y el segundo predicado es Verdadero
-- y en este caso me deberia dar  a los tres clientes con la f 50
-- por lo tanto este predicado es una tautologia y siempre sera verdadero.
-- Sin embargo SQL nos devuelve un valor  que no se corresponde con el mundo real
```

Clase 10 y 12

14. Teoría del Diseño Relacional

15. Forma Normal

Son reglas para determinar si un esquema que hicimos es lo suficientemente prolijo.

Para que un modelo sea correcto:

- Tiene que preservar la información, todo lo hecho en el diseño conceptual este en el relacional.
- Tiene que tener redundancia mínima, que no hagamos la misma cosa mas de una vez. (Por ejemplo, código postal y localidad, con el código postal ya podemos tener el otro). La redundancia no solo ocupa mas espacio, lleva a inconsistencias.

Cuando se hace un correcto pasaje, se tiene un esquema sin redundancia y preserva la información que queríamos. Para saber si esto lo hicimos bien necesitamos las formas normales.

Dependencias funcionales

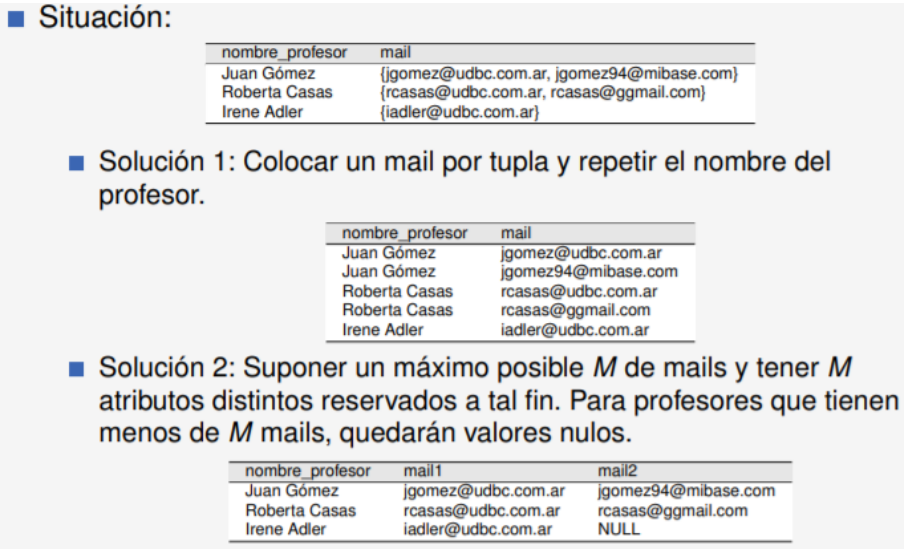
Dada una relación, podemos identificar una dependencia funcional como una restricción de que un atributo determina otro $X \to Y$. *Dos tuplas con igual código postal deben de tener la misma localidad. (localidad es función de código postal) (RIGE esta dependencia) (No es cierto del otro lado).*

Para trabajar con esto, se hace conceptualmente, tenemos que pensar que es lo que representan los datos de mi relación. NO tenemos que guiarnos por los datos que tenemos.

- Cuando Y esta incluido en X, decimos que Y es trivial.
- Las dependencias funcionales siempre se definen a partir de la semántica de los datos, no es posible inferirla con los datos.
- Una dependencia que esta bien es legajo implica apellido, CP, localidad (surge a partir de la clave primaria/candidatas).
- Hay varias formas normales, si una cumple con una, cumplen con las anteriores. (son subconjuntos) (*Si esta en 3FN esta en 2FN, si esta en 4FN esta en 3FN, ... incluyen a las anteriores.*)
- El proceso de normalización es una forma de pasar de una forma normal a una superior que preserva toda la información. Partimos de un conjunto de dependencias funcionales que supondremos definido por el diseñador de la base de datos.

Primera forma normal

Los dominios de todos sus atributos solo permiten valores atómicos y monovaluados. En el modelo relacional todos deben de ser así, por lo que no es algo que se vea en las bases de datos. Se resolvería creando una tabla separando.



Segunda forma normal

Cuando una dependencia solo depende de parte de una clave primaria (Siempre ver las claves candidatas), decimos que es una dependencia funcional parcial. Buscamos que no haya de estas.

$X \to Y$ es parcial cuando existe un subconjunto propio A incluido en X, con A distinto de X para el cual A implica Y. Una dependencia funcional $X \to Y$ es completa si y sólo si no es parcial.

Atributo primo de una relación: es aquel que es parte de una clave candidata de la relación

---

Decimos que una relación esta en 2FN cuando todos sus atributos no primos dependen por completo de las claves candidatas. Cuando no es así, solucionamos esto realizando otra tabla, separamos lo que participa en una dependencia llevándola a otra.

1. Identificamos las claves candidatas.
2. Clasificamos a los atributos en primos y no primos.
3. Revisamos dependencias y resolvemos.

Descomposición: Es tener una relación y partirla en varias mas. Es una descomposición de la relación cuando se preserva la información.

Si una descomposición cumple que para toda instancia posible de R, la junta de las proyecciones sobre los R_i permite recuperar la misma instancia de relación, entonces decimos que la descomposición preserva la información.

Diremos que la descomposición preserva las dependencias funcionales cuando toda dependencia funcional $X \to Y$ en R puede inferirse a partir de dependencias funcionales definidas en los R_i.

Una descomposición de R que cumple con ambas propiedades se denomina descomposición equivalente de R.

TENIS

nombre_torneo	año	ciudad	país	tenista1	tenista2	ronda	set	punt1	punt2
Roland Garros	2016	París	Francia	A. Murray	S. Wawrinka	2-final	1	6	4
Roland Garros	2016	París	Francia	A. Murray	S. Wawrinka	2-final	2	6	2
Roland Garros	2016	París	Francia	A. Murray	S. Wawrinka	2-final	3	4	6
Roland Garros	2016	París	Francia	A. Murray	S. Wawrinka	2-final	4	6	2
Masters de Madrid	2015	Madrid	España	R. Federer	R. Nadal	4-final	1	3	6
Masters de Madrid	2015	Madrid	España	R. Federer	R. Nadal	4-final	2	1	6
Roland Garros	2016	París	Francia	N. Djokovic	A. Murray	Final	1	6	3
Roland Garros	2016	París	Francia	N. Djokovic	A. Murray	Final	2	1	6
Roland Garros	2016	París	Francia	N. Djokovic	A. Murray	Final	3	6	2
Roland Garros	2016	París	Francia	N. Djokovic	A. Murray	Final	4	6	4

Hipótesis: Todos los torneos son por eliminación, de manera que 2 tenistas pueden enfrentarse 1 vez como máximo por torneo.

- Identificamos las dependencias funcionales no triviales a partir de la semántica:
  - nombre_torneo → {ciudad, país}
  - {nombre_torneo, año, tenista1, tenista2} → {ronda}
  - {nombre_torneo, año, tenista1, ronda} → {tenista2}
  - {nombre_torneo, año, tenista2, ronda} → {tenista1}
  - {nombre_torneo, año, tenista1, tenista2, set} → {punt1, punt2}
  - {nombre_torneo, año, tenista1, ronda, set} → {punt1, punt2}
  - {nombre_torneo, año, tenista2, ronda, set} → {punt1, punt2}

- Identificamos la clave primaria
  - {nombre_torneo, año, tenista1, tenista2, set}
- Identificamos otras claves candidatas
  - {nombre_torneo, año, tenista1, ronda, set}
  - {nombre_torneo, año, tenista2, ronda, set}
- ¿Dependencias funcionales parciales de una clave candidata de atributos no primos?
  - nombre_torneo → {ciudad, país}

TORNEOS

nombre_torneo	ciudad	país
Roland Garros	París	Francia
Masters de Madrid	Madrid	España

PARTIDOS

nombre_torneo	año	tenista1	tenista2	ronda	set	punt1	punt2
Roland Garros	2016	A. Murray	S. Wawrinka	2-final	1	6	4
Roland Garros	2016	A. Murray	S. Wawrinka	2-final	2	6	2
Roland Garros	2016	A. Murray	S. Wawrinka	2-final	3	4	6
Roland Garros	2016	A. Murray	S. Wawrinka	2-final	4	6	2
Masters de Madrid	2015	R. Federer	R. Nadal	4-final	1	3	6
Masters de Madrid	2015	R. Federer	R. Nadal	4-final	2	1	6
Roland Garros	2016	N. Djokovic	A. Murray	Final	1	6	3
Roland Garros	2016	N. Djokovic	A. Murray	Final	2	1	6
Roland Garros	2016	N. Djokovic	A. Murray	Final	3	6	2
Roland Garros	2016	N. Djokovic	A. Murray	Final	4	6	4

Tercera Forma Normal

- No se esta en 3FN cuando hay un atributo no primo que implica otro atributo no primo y no es trivial. Hay una dependencia transitiva de la clave candidata.
- Separamos a Z y a Y en una nueva relación dejando en la relación original al Z. (Nuevas tablas)
- Toda dependencia funcional parcial no trivial es transitiva.
- Se esta en 3FN cuando no existen dependencias transitivas de atributos no primos.

- Otra definición es: para toda dependencia funcional no trivial $X \rightarrow Y$, o bien X es superclave, o bien Y - X contiene sólo atributos primos.

DETALLEFACTURA

nro_factura	nro_item	cod_producto	nombre_producto	cantidad	precio_unit
0003-45821	1	249	Suprabond 500mg	2	87.00
0003-45821	2	230	Tersuave azul 4l	1	270.00
0003-45821	3	115	Brocha 5cm	2	90.00
0003-45822	1	258	Alba p/Exteriores 3l	2	225.00
0003-45822	2	116	Brocha 10cm	2	130.00
0003-45823	1	330	Cetol 2l	1	315.00

PK = {*nro_factura*, *nro_item*}

¿Cómo se resuelve la situación?

- DetalleFactura(nro_factura nro_item cod_producto cantidad precio_unit)
- Productos(cod_producto, nombre_producto)

Forma Normal Boyce-Codd

- La FNBC dice que no quiere ver ninguna dependencia transitiva (no importa si es primo)
- La parte izquierda de toda dependencia funcional no trivial debe de ser superclave. Se arregla separando en una tabla.
- Esta forma resuelve casos en los que hay varias clases candidatas, pero pierde dependencias funcionales. Es un compromiso entre la mínima redundancia y perder dependencias funcionales (no siempre se pierden).

PARTIDOS

nombre_torneo	año	tenista1	tenista2	ronda	set	punt1	punt2
Roland Garros	2016	A. Murray	S. Wawrinka	2-final	1	6	4
Roland Garros	2016	A. Murray	S. Wawrinka	2-final	2	6	2
Roland Garros	2016	A. Murray	S. Wawrinka	2-final	3	4	6
Roland Garros	2016	A. Murray	S. Wawrinka	2-final	4	6	2
Masters de Madrid	2015	R. Federer	R. Nadal	4-final	1	3	6
Masters de Madrid	2015	R. Federer	R. Nadal	4-final	2	1	6
Roland Garros	2016	N. Djokovic	A. Murray	Final	1	6	3
Roland Garros	2016	N. Djokovic	A. Murray	Final	2	1	6
Roland Garros	2016	N. Djokovic	A. Murray	Final	3	6	2
Roland Garros	2016	N. Djokovic	A. Murray	Final	4	6	4

La "ronda" puede deducirse con sólo una parte de la clave primaria, y sin embargo la estamos repitiendo en cada set.

La FNBC impide que esto suceda prohibiendo que existan dependencias parciales de una clave candidata, inclusive de atributos primos.

Una dependencia que nos molesta es {nombre_torneo, año, tenista1, tenista2}→ *ronda*, porque {nombre_torneo, año, tenista1, tenista2} no es superclave.

Clausura de F

Es el conjunto formado por todas las dependencias funcionales de F. No tiene sentido calcularla, sirve mas para entender el concepto.

Queremos trabajar con el conjunto que tenga menos. (llamado cubrimiento mínimal del conjunto de dependencias) Equivalente y con menos elementos.

Todo lo que se puede generar con la clausura uniendo dos relaciones es lo que se preserva. De esta forma podemos ver si perdimos algo.

Cuando vemos una dependencia funcional que no cumple los requisitos en una relación R, podemos separarla.

- $R(A)$
- $X \to Y$
- $R_1(A - (Y - X))$
- $R_2(X \cup Y)$

Se garantiza que la junta de las proyecciones R1 y R2 recupera la R original.

Cuando una descomposición pierde información, no es que se pierden tuplas, lo que sucede es que se crean nuevas.

Dependencia multivaluada

$X \twoheadrightarrow Y$ es una restricción sobre las posibles tuplas de R que implica que para todo par de tuplas t1, t2 tales que t1[x] = t2[x] deberían existir otras dos tuplas t2 y t4 que resulten de intercambiar los valores de Y entre t1 y t2. (Implican otras tuplas)

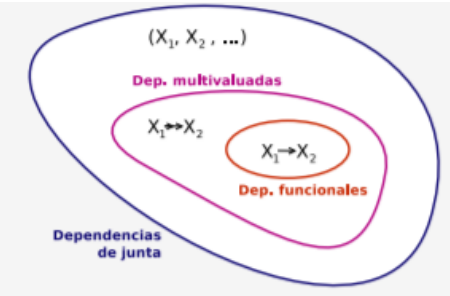
Son triviales si la unión de X e Y da la misma tabla o Y esta incluido en X.

Cuarta Forma Normal

No entra en el parcial, entra en el final.

No tiene que haber dependencias multivaluadas no triviales $X \twoheadrightarrow Y$. Se esta en 4FN si para toda $X \twoheadrightarrow Y$, X es superclave.

- Hay que identificar la dependencia multivaluada. No es fácil con una tabla ya armada. Si se hace el paso desde el modelo conceptual de forma correcta no aparecen estos problemas.



- Si R esta en 4FN, entonces R esta en FNBC
- Es común que las dependencias multivaluadas provengan de la existencia de atributos multivaluados en el modelo conceptual, o de interrelaciones N-N no capturadas.

Descomponemos generando nuevas tablas.

PROMOCIONESVENDIDAS

nro_factura	nombre_cliente	descripción_promo	nombre_producto
0249-19855	Juana Auzqui	Fiesta-Pancho	Pack salchichas x6
0249-19855	Juana Auzqui	Fiesta-Pancho	Pack pan de viena x6
0249-19855	Juana Auzqui	Fiesta-Pancho	Mayonesa 250gr
0034-20329	Bernardo Lühn	Vajilla Reluciente	Esponjas x2
0034-20329	Bernardo Lühn	Vajilla Reluciente	1 detergente Universo
0034-20329	Bernardo Lühn	Vajilla Reluciente	1 antigrasa Universo
0034-20329	Bernardo Lühn	Vajilla Reluciente	Repasadores x3
0058-91330	Bernardo Lühn	Fiesta-Pancho	Pack salchichas x6
0058-91330	Bernardo Lühn	Fiesta-Pancho	Pack pan de viena x6
0058-91330	Bernardo Lühn	Fiesta-Pancho	Mayonesa 250gr

Clave de la relación:

- {*nro_factura*, *descripcion_promo*, *nombre_producto*}.

No es cierto que "nombre_producto" dependa funcionalmente de "descripción_promo". Sin embargo, tenemos información redundante porque *los productos que integran cada promo son siempre los mismos* ***independientemente*** *de quienes compran la promo*.

Primero normalizamos para llevar a FNBC eliminando la dependencia funcional parcial *nro_factura* → *nombre_cliente*. Para ello descomponemos en:

Descomposición a FNBC

ClienteFactura(nro_factura, nombre_cliente)
PromoProdFactura(nro_factura, descripción_promo, nombre_producto)

Luego eliminamos la dependencia multivaluada *descripcion_promo* ↠ *nombre_producto* descomponiendo en:

Descomposición a 4FN

Promociones(descripción_promo, nombre_producto)
ClientesFactura(nro_factura, nombre_cliente)
PromocionesFactura(nro_factura, descripción_promo)

PROMOCIONES

descripción_promo	nombre_producto
Fiesta-Pancho	Pack salchichas x6
Fiesta-Pancho	Pack pan de viena x6
Fiesta-Pancho	Mayonesa 250gr
Vajilla Reluciente	Esponjas x2
Vajilla Reluciente	1 detergente Universo
Vajilla Reluciente	1 antigrasa Universo
Vajilla Reluciente	Repasadores x3

CLIENTESFACTURA

nro_factura	nombre_cliente
0249-19855	Juana Auzqui
0034-20329	Bernardo Lühn

PROMOCIONESFACTURA

nro_factura	descripción_promo
0249-19855	Fiesta-Pancho
0034-20329	Vajilla Reluciente
0058-91330	Fiesta-Pancho

Dependencias de junta

Es cuando tenemos relaciones que pueden ser descompuestas en más de dos relaciones sin perdida de información.

Quinta Forma Normal

No entra en el parcial, entra en el final.

- Una relación R(A) está en quinta forma normal (5FN) si y sólo si
- para toda dependencia de junta (X1, X2, ..., Xn) no trivial (i.e., tal
- que ningún Xi = A) todos los Xi son superclaves.
- Se descompone realizando una tabla por cada relación.
- Es muy difícil detectar dependencias de junta en forma general, y
- esta descomposición rara vez es aplicada

Un supermercado tiene varias sucursales que comercializan distintos tipos de productos (p.ej., lácteos, vinos, elementos de bazar, yerbas, etc). El supermercado trabaja con distintos proveedores y no todos comercializan todo. Pero cuando una sucursal trabaja con un cierto proveedor, le adquiere todos los productos que la sucursal comercializa y el proveedor ofrece.

COMERCIALIZACIÓN

sucursal	proveedor	tipo_producto
Floresta	El Picadero	Vinos
Floresta	La Bondad	Leches
Floresta	La Bondad	Yerbas
La Boca	Blanquín	Bazar
La Boca	Blanquín	Pañales
La Boca	Pirulo	Bazar
Villa del Parque	El Picadero	Vinos
Villa del Parque	El Picadero	Quesos
Villa del Parque	Blanquín	Bazar
Recoleta	Blanquín	Bazar
Recoleta	Romualdo	Quesos
Recoleta	El Picadero	Vinos
Recoleta	El Picadero	Quesos

PROVEEDORES SUCURSALES

sucursal	proveedor
Floresta	El Picadero
Floresta	La Bondad
La Boca	Blanquín
La Boca	Pirulo
Villa del Parque	El Picadero
Villa del Parque	Blanquín
Recoleta	Blanquín
Recoleta	Romualdo
Recoleta	El Picadero

PRODUCTOS PROVEEDORES

proveedor	tipo_producto
El Picadero	Vinos
La Bondad	Leches
La Bondad	Yerbas
Blanquín	Bazar
Blanquín	Pañales
Pirulo	Bazar
El Picadero	Quesos
Romualdo	Quesos

PRODUCTOS SUCURSALES

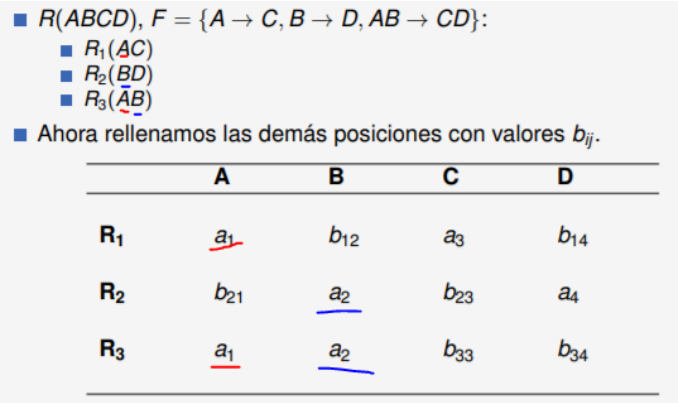
sucursal	tipo_producto
Floresta	Vinos
Floresta	Leches
Floresta	Yerbas
La Boca	Bazar
La Boca	Pañales
Villa del Parque	Vinos
Villa del Parque	Quesos
Villa del Parque	Bazar
Recoleta	Quesos
Recoleta	Vinos

→ Existe la siguiente dependencia de junta:
({sucursal, proveedor}, {proveedor, tipo_producto}, {sucursal, tipo_producto}).

Algoritmo de verificación de junta sin perdidas

- Se llama algoritmo de Chase. Nos permite verificar la preservación de información de una descomposición aun sin saber como se obtuvo.
- El algoritmo chase utiliza una tabla denominada tableau, con
- tantas filas como relaciones y tantas columnas como atributos
- El algoritmo parte de una hipotética tupla (a1, a2, a3, a4) de la
- junta r1 ⋈ r2 ⋈ r3 que se proyecta a cada una de las ri: si una
- relación ri contiene un atributo Aj, entonces en la posición (i, j) de
- la tabla escribimos el valor abstracto aj. (las filas son las relaciones, las columnas los atributos) (las aj van en los lugares que comparten atributos las relaciones)

- Rellenamos las demás posiciones con valores bij.



- $R(ABCD)$, $F = \{A \to C, B \to D, AB \to CD\}$:
  - $R_1(AC)$
  - $R_2(BD)$
  - $R_3(AB)$
- Ahora rellenamos las demás posiciones con valores b_{ij}.

	A	B	C	D
R1	a_1	b_{12}	a_3	b_{14}
R2	b_{21}	a_2	b_{23}	a_4
R3	a_1	a_2	b_{33}	b_{34}

- Después de rellenada, vamos remplazando los valores con las dependencias funcionales. (Priorizando elegir las a)
- Cuando llegamos al final y no nos quedo alguna fila con todas a, decimos que la descomposición no preserva la información.
- Cuando no funciona el Chase nos da un contraejemplo mostrando donde la relación falla.
- Si llegamos al final y nos da una fila con todas a, preserva la información.

Clase 11

16. Practica: Normalización

- Tomar un esquema y lo partimos en partes más pequeñas para llegar a esquemas más depurados.
- La creación de tuplas nuevas al unir tablas con datos apócrifos hace que se pierda toda la información.
- En esquemas defectuosos hay ambigüedades, se pierde información, existen valores nulos.
- El esquema conceptual nos da una herramienta valiosa para modelar. Cuando hacemos el mapeo al modelo lógico buscamos obtener un buen esquema.
- "Hechos distintos se deben almacenar en objetos distintos."
- Una dependencia funcional es una restricción sobre el conjunto de tuplas que pueden aparecer en una relación.
- Tenemos que preguntar sobre los datos. ¿Que significa este código? ¿Este sector? ¿Que depende de que?

¿Como se descubren las dependencias funcionales?

- No podemos inferirlas a partir de una tabla. Necesitamos de un relato que nos brindara el conocedor del dominio del problema. Esto es fundamental para diseñar correctamente la base de datos que responda al problema planteado. (Tomamos cada frase y la analizamos)
- Cuando dos elementos se están asociando entre si, digo que son equivalentes. $X \to Y; Y \to X$
- La clave y superclave son casos particulares de dependencias funcionales. Decimos que X es una clave si X implica todos los atributos (R - la relación) y no existe ningún Z tal que este contenido en X e implique R (debe de ser minimal).

¿Como operamos con ellas?

Usamos los Axiomas de Armstrong (RAT). Con esto se pueden descubrir dependencias que por ahí están ocultas en otras.

Usar las definiciones y se va a llegar al resultado.

- **Reflexividad** Si Y esta incluido en X, podemos inferir que X implica Y. Si $Y \subseteq X$ inferir $X \to Y$
- **Aumento**: Para cualquier conjunto W, de X a Y puedo inferir XW implica YW. $X \to Y$ inferimos $XW \to YW$
- **Transitividad**: De X implica Y e Y implica Z podemos inferir que X implica Z. $X \to Y$ e $Y \to Z$ inferimos $X \to Z$

Clausura de F (F^+)

Dado un conjunto de dependencias funcionales de F, la clausura $F+$ es el conjunto de todas las dependencias funcionales que pueden inferirse con los axiomas de Armstrong.

$$F+ = \{X \to Y / F| = X \to Y\}$$

- $F| = X \to Y$ significa que esa dependencia funcional esta implicada por F

Clausura de un conjunto de atributos X_F^+

Es el conjunto de todos los atributos contenidos en la relacion tal que X implica esos atributos derivado con los axiomas de Armstrong (Con X un atributo).

$$X_F^+ = \{A, A \subseteq R \ y \ F| = X \rightarrow A\}$$

Equivalencias de conjuntos

Dos conjuntos de dependencias son equivalentes si sus clausuras son iguales. $F^+ = G^+$

Esto nos permite tener herramientas para transformar conjuntos en otros con propiedades mas deseables.

Conjunto minimal F_M

- Todo implicado de una dependencia funcional (parte derecha) tiene un único atributo (es simple)
- Todo determinante (parte izquierda) de una dependencia funcional es reducido (no contiene atributos redundantes)
- F_M no contiene dependencias redundantes.

Algoritmo

1. Dejamos todos los lados derechos con un único atributo (*Ej.* $A \rightarrow BD$ *se transforma en* $A \rightarrow B$, $A \rightarrow D$)
2. Eliminamos todos los atributos redundantes del lado izquierdo (*Ej. Analizo* $AC \rightarrow E$, *por* $A \rightarrow B$ *y* $B \rightarrow C$ *infiero* $A \rightarrow C$ *entonces* $A \rightarrow E$ - *O lo que es lo mismo veo si* $E \subset A^+ = \{A, B, C, D, E\}$ *(sin la dependencia que estoy analizando), si pertenece a la clausura, no necesito al C (puede ser a la inversa tambien)*)
3. Eliminamos las dependencias funcionales redundantes. Vemos si están en la clausura del conjunto de atributos y restamos esta del conjunto si la incluye. $X \rightarrow Y si Y \subset X^+_{FD2-\{X \rightarrow Y\}}$ Si no incluye al atributo NO es redundante. Si alguna es redundante, se continua el análisis sin esa dependencia, se resta del conjunto FD2 junto a la dependencia que estamos analizando. *Ej.* $FD2 - \{DepRedundante\} - \{X \rightarrow Y\}$ (Si X son atributos compuestos los veo juntos)

Dependiendo del orden de como procese las cosas pueden quedar más de un conjunto minimal. Son equivalentes igual.

Algoritmo de calculo de claves

1. Calcular el cubrimiento minimal de dependencias funcionales del conjunto de atributos para calculo = Ca = R (el conjunto minimal de F)
2. Detectar atributos independientes del calculo Ai (no están en ninguna dependencia funcional), eliminar del conjunto Ca=Ca-Ai y reservar para después
3. Eliminar atributos equivalentes (dejar solo uno) Ae = atributos equivalentes menos uno. Ca=Ca - Ae (reservarlos para otro paso). Se remplazan en la dependencia funcional por el los atributos equivalentes que se conservan en Ca
4. Se forma K con todos los elementos que sean solo implicantes Ai (solo en parte izquierda), se calcula K^+ si es todo R =>K es clave

5. Si K no resulto clave, se busca el conjunto de elementos que estén entre los implicantes pero que puedan ser implicados Aid (están en parte derecha e izquierda). Se agregan alternativamente a K todos los posibles subconjuntos de Aid (todos los de 1 elementos, 2 elementos, etc). Se verifica que K sea clave (calculando la clausura), en este paso se deben obviar los subconjuntos que contienen una clave ya calculada, ya que no van a ser minimales.
6. Se agregan a la clave los elementos independientes
7. Se calculan las otras claves con los elementos equivalentes

Los elementos a la derecha nunca forman parte de la clave.

Transformación por proyecciones

Son equivalentes los esquemas si se conserva la información y se conservan las dependencias funcionales.

Tipos de atributos

- Atributos primos son los que pertenecen a alguna clave candidata.
- Los no primos no pertenecen a ninguna.

Formas normales

- 2FN ningún atributo no primo depende parcialmente de alguna clave.
- 3FN En toda dependencia no trivial $X \to A$ perteneciente a F, al lado izquierdo es siempre una superclave o A son atributos primos.
- FNBC En toda dependencia no trivial $X \to A$ perteneciente a F, al lado izquierdo es siempre superclave.

Superclave es que contiene alguna clave candidata y algún elemento de mas.
Una dependencia la puedo perder, información nunca.

Clase 13

17. Practica: Normalización II

Algoritmo de descomposición en 3FN

Este algoritmo es constructivo, garantiza la preservación de dependencias y la información (por incluir una clave del esquema original) Entra un esquema R y salen varios Ri en 3FN que preservan información y dependencias funcionales.

1. Buscamos un conjunto minimal para F
2. Encontramos las claves de R
3. Para cada dependencia funcional $X \to A_i$ en F_{min} creamos un esquema que tenga a X y a A_i.
4. Si ningún esquema de los generados contiene una clave de R, se crea un esquema adicional que contenga atributos que formen una superclave de R. Necesitamos encontrar por lo menos una en las relaciones resultantes.
5. Opcional, pero bueno, es unir los esquemas que tengan la misma clave primaria.

Un problema del algoritmo es que puedo fragmentar demasiado, tener muchos esquemas chicos.

Algoritmo de descomposición de FNBC

Tiene la propiedad NJB (comprobación de concatenación no aditiva para descomposiciones binarias) si y solo si la df entre las relaciones R1 y R2 implican la resta de R1 con R2, o R2 con R1 esta en la clausura transitiva de F.

Cuando descomponemos puede pasar que perdamos df. Se garantiza que se mantiene la información por propiedad NJB.

Entra un esquema R y salen varios Ri en FNBC que preservan información.

Vemos en que FN esta, y ahi decidimos si corresponde aplicar el algoritmo.

1. Establecemos D=$\{R\}$
2. Mientras que exista un esquema Q en D que no este en FNBC
   a) Escogemos Q en D que no este en FNBC
   b) Encontrar una dependencia funcional $X \to Y que viole FNBC$
   c) Reemplazamos Q en D por dos esquemas. Un esquema es $R_1 = \{X\}^+ S_1 = proyeccin Sen R_1$ y el otro $R2 = \{Q - \{X\}^+ \cup X\} S_2 = proyeccin Sen R_2$
3. Procesar recursivamente R1 y R2

El resultado es que se va formando un árbol. Este conviene dibujarlo para no perderse.

Cuando descomponemos puede que perdamos df's, pero también puede que en los nuevos esquemas se puedan representar otras dependencias que no estén explicitadas en nuestro conjunto de df´s, pero que implícitamente existan. Entonces debemos proyectar nuestro conjunto de df´s en el nuevo conjunto.

Algoritmo Tableau

Detecta perdida de información. Lo hace mapeando todas las descomposiciones de R y luego se va operando con las dependencias funcionales para tratar de reconstruir la relación universal. Si podemos reconstruirla no hubo perdida de información. Este método es una simplificación del método Chase

- Entra una relación, sale un si o un no.
- Construimos una matriz con NFIL y NCOL, la columna k corresponde al atributo Aj y la fila i corresponde al esquema Ri de PROY. Los valores de las columnas, para cada una de las filas Ri de la matriz, se rellenan de la siguiente forma:
  - Si Aj está en Ri, MATij=Aj
  - Si Aj no está en Ri, MATij=biAj

Luego vamos aplicando las dependencias funcionales. Tenemos que lograr que una fila con todos los atributos.

Donde participa cada atributo en las relaciones, pongo las letras correspondientes a esos atributos en la matriz, el resto lo relleno con bij.

Aplicamos dependencias, donde tengo mismos valores, tiene que implicar lo mismo, remplazando los valores (priorizamos las mayúsculas). (Queremos lograr una fila con letras mayúsculas) (Si la implicación tiene mas de un atributo de un lado, buscamos esa cantidad, esos pares tienen que ser iguales)

No importa el orden en que apliquemos las dependencias, es recursivo, ante cualquier cambio hay que volver a aplicar todas las dependencias.

Clase 14

18. Concurrencia y Transacciones

El interés de la concurrencia surge de aprovechar la capacidad de procesamiento lo mejor posible para atender a los usuarios. Que no estén en una cola de espera.

Tenemos sistemas monoprocesador que permiten hacer multitasking (varios hilos o procesos) y sistemas multiprocesador o distribuidos (varias unidades de procesamiento o nodos que replican la base de datos en distintas unidades de procesamiento). Aun con multiprocesador la concurrencia nos puede hacer ganar. Se mejora el response time al solapar las tareas.

Transacción es una unidad lógica de trabajo compuesta por una secuencia de instrucciones atómicas (no se pueden dividir). Pueden ser operaciones de consulta o ABM. Queremos que se ejecuten por completo o que no se ejecuten. No queremos que queden por la mitad.

La **concurrencia** es la posibilidad de ejecutar múltiples transacciones en forma simultanea (tareas). Vamos a querer aprovechar toda la capacidad de computo. El problema que genera la ejecución concurrente es la gestión de los recursos compartidos. Al nivel de los SGBDs los recursos compartidos son los datos, a los cuales distintas transacciones quieren acceder en forma simultanea.

El modelo de procesamiento que usaremos es el de concurrencia solapada que considera que:

- Hay un solo procesador.
- Cada transacción esta formada por instrucciones atómicas.
- El scheduler puede suspender en cualquier momento las instrucciones.

Item: Puede representar el valor de un atributo en una fila determinada de una tabla, una fila de una tabla, un bloque del disco, una tabla. El tamaño de este se conoce como **granularidad**, afecta sustancialmente al control de concurrencia.

Las instrucciones atómicas serian:

- leer_item(X) lee el valor de X cargándolo en memoria
- escribir_item(X) ordena escribir el valor que esta en memoria del item X en la base de datos

Hay que tener en cuenta que:

- En el medio de las instrucciones se realizan otras operaciones en el CPU que no nos afectan al análisis de concurrencia. (Por ejemplo una junta en memoria)
- Ordenar escribir no necesariamente lleva a disco. (puede quedar en un buffer)

Propiedades ACID

El gestor tiene que garantizar estas propiedades en todo momento.

- **Atomicidad**: Las transacciones deben ejecutarse de forma atómica, se ejecutan o no. (El usuario las ve de esta forma) (se garantiza con el log)
- **Consistencia**: Cada ejecución debe preservar la consistencia de los datos. Se define con reglas de integridad (se deben de verificar en todo momento, ej. no puede haber mas de un gerente por departamento).

- **Aislamiento**: El resultado de la ejecución concurrente de las transacciones debe ser el mismo que si las transacciones se ejecutaran en forma aislada una tras otra, es decir en **forma serial**. La ejecución concurrente debe entonces ser equivalente a alguna ejecución serial. Lo que percibe el usuario de datos. Si el gestor cumple con el aislamiento y miro un valor en la base de datos con un solapamiento, este debe de tener el mismo valor si fuera serial la ejecución (una transacción después de otra) (en alguno de todos los ordenes posibles que hay) (se garantiza con el log)
- **Durabilidad**: Una vez que el SGBD informa que la transacción se completo, debe garantizarse la persistencia de la misma independientemente de toda falla que pueda ocurrir. (se garantiza con el log, deshace o rehace con lo que le dice)

Las propiedades se garantizan con mecanismos de recuperación. Buscan garantizar la visión de todo o nada de las transacciones. Se agregan algunas instrucciones especiales

- **begin**: se comenzó la transacción.
- **commit**: se termino con éxito.
- **abort**: ocurrió un error y todos los efectos de la transacción deben ser deshechos. (rollback)

Estos mecanismos de recuperación se van registrando en un archivo. Esto permite cumplir ACID. Si ocurre una falla, el log le dice como volver atrás. Siempre se tiene que escribir a disco, se suele usar uno especial de acceso rápido que se escribe en forma secuencial. No es lo mismo que estar escribiendo a la base de datos en disco. Cada cierto tiempo este log se va limpiando.

Anomalías

Son situaciones que pueden violar ACID. Les decimos anomalía cuando ya no se puede arreglar, si se puede arreglar lo que paso se llama fenómeno.

Dirty read

- Sucede cuando una transacción lee un valor modificada por otra transacción **que aun no se commiteo**. Se conoce también como Read uncommitted data. Es un conflicto de escritura lectura (WR). (Si aborta la transaccion que lo modifico al dato se produce la anomalia)
- No tiene solución una vez que pasa, arruina la consistencia de la base de datos.
- La forma de evitarlo es no permitir que la transacción haga commit hasta que la otra haga abort o commit. La otra opción es que no lea, pero es mas drástica esta solución.

Actualización perdida - lost update

- Alguien escribió lo que otro ya había leído, deriva en una anomalía si después el primero vuelve a escribir (lost update) o vuelve a leer (lectura no repetible).
- Se pisa una modificación con algo ya leído. Si la primera transacción luego modifica y escribe lo que se leyó y se pierde por otra transacción.
- Puede pasar que esa misma transacción vuelva a leer el mismo item y tiene algo distinto, causando una rotura del aislamiento (no hay ningún orden serial en el cual pueda pasar esto). (unrepeatable read)
- Ambas situaciones se conocen como RW (read write) seguido por otro de tipo WW o WR.
- Ej. Una transaccion deposita 100$ y otra retira $100, en este caso el cambio debería de ser 0, pero puede pasar que aparezcan -100 o +100.
- Rompe el aislamiento, no tiene un orden serial, no es serializable.

Dirty write

Ocurre cuando una transacción T2 escribe un item que ya había sido escrito por otra transacción T1 que luego se deshace. Se conoce como WW (write-write) o Overwrite uncommited. El gestor cuando aborte va a intentar poner el valor que habia antes, pisando lo que hizo otro.

Phantom

- *Aparecen y desaparecen cosas en la base de datos*
- Transacción T1 que observa un conjunto de items que cumplen una condición y luego el conjunto cambia porque algunos de sus items fueron modificados/creados/eliminados. Si esto sucede mientras T1 se esta ejecutando podría encontrarse con un conjunto que cambio.
- Si está modificación se hace mientras T1 aún se está ejecutando, T1 podría encontrarse con que el conjunto de ítems que cumplen la condición cambió.
- Atenta contra la serializabilidad. Para evitarlo es necesario usar locks a nivel de tabla o predicado.

Notación

Para analizar la serializabilidad de un conjunto de transacciones en nuestro modelo de concurrencia solapada, utilizaremos la siguiente notación breve para las instrucciones:

- $R_T(X)$: La transacción T lee el ítem X.
- $W_T(X)$: La transacción T escribe el ítem X.
- b_T: Comienzo de la transacción T.
- c_T: La transacción T realiza el *commit*.
- a_T: Se aborta la transacción T *(abort)*.

Con esta notación, podemos escribir una transacción general T como una lista de instrucciones $\{I_T^1; I_T^2; ...; I_T^{m(T)}\}$, en donde $m(T)$ representa la cantidad de instrucciones de T.

Ejemplo:

- $T_1 : b_{T_1}; R_{T_1}(X); R_{T_1}(Y); W_{T_1}(Y); c_{T_1};$
- $T_2 : b_{T_2}; R_{T_2}(X); W_{T_2}(X); c_{T_2};$

- No nos importan las operaciones que realiza en memoria.
- Un **solapamiento** entre dos transacciones T1 y T2 es una lista de m(T1) + m(T2) instrucciones, en donde cada instrucción de T1 y T2 aparece una única vez, y las instrucciones de cada transacción conservan el orden entre ellas dentro del solapamiento.
- Cantidad de solapamientos posibles: $\frac{(m(T_1)+m(T_2))!}{m(T_1)!\ m(T_2)!}$
- Nos interesa ver si el solapamiento es serializable

Ejecución serial es aquella en que las transacciones se ejecutan por completo una detrás de otra en base a algún orden. Existen $n!$ ejecuciones seriales distintas.

Decimos que un solapamiento de un conjunto de transacciones T1, T2, ..., Tn es **serializable** cuando la ejecución de sus instrucciones en dicho orden deja a la base de datos en un estado

equivalente a aquél en que la hubiera dejado alguna ejecución serial de T1, T2, ..., Tn. Nos interesa esto porque garantizan la propiedad de aislamiento de las transacciones.

Equivalencia

- Por **resultados**, mi ejecución de solapamiento da lo mismo **para un estado inicial particular**. Cuando, dado un estado inicial particular, ambos órdenes de ejecución dejan a la base de datos en el mismo estado. (Es una equivalencia mas floja)
- Por **conflictos**, implica la de por resultados (no al revés), no depende del estado inicial. Cuando ambos órdenes de ejecución poseen los mismos conflictos entre instrucciones. (Da lo mismo, es serializable independientemente de los valores) (El solapado como el serial tienen los mismos conflictos) **No dependen del estado inicial**
- De **vistas**, es una intermedia, más débil que la de conflictos, más fuerte que la de resultados. Cuando en cada orden de ejecución, cada lectura RTi(X) lee el valor escrito por la misma transacción j, WT(X). Además se pide que en ambos órdenes la última modificación de cada ítem X haya sido hecha por la misma transacción.

Conflicto

Dado un orden de ejecución, un conflicto es un par de instrucciones (I1, I2) ejecutadas por dos transacciones distintas Ti y Tj, tales que I2 se encuentra más tarde que I1 en el orden, y que responde a alguno de los siguientes esquemas:

- una transacción escribe un item que otra leyó. (Ri, Wj)
- una transacción lee un item que otra escribió. (Wi, Rj)
- dos transacciones escriben un mismo item. (Wi, Wj)

Todo par de instrucciones consecutivas (I1, I2) de un solapamiento con transacciones distintas que no constituye un conflicto puede ser invertido en su ejecución. (Tienen que estar pegadas las instrucciones) Haciendo este swap podemos llegar a el orden serial.

Grafo de precedencias

- Nos dice si hay conflictos, no los evita.
- Queremos ir swappeando y llegar a que quede toda la transacción 1 a la izquierda.
- Buscamos evaluar si un solapamiento es serializable o no.
- Los nodos son transacciones y se agrega un arco entre los nodos i,j si existe algún conflicto de los mencionados. (WR,RW,WW)
- Opcionalmente se etiqueta el arco con el item que causa el conflicto.
- Si el grafo tiene ciclos, no es serializable.
- Un orden de ejecución es serializable por conflictos si y solo si su grafo de precedencias no tiene ciclos.
- Algoritmo de ordenamiento topológico si tengo un grafo acíclico y quiero el orden.

Control de concurrencia

Tenemos dos formas:

- Enfoque pesimista, busca garantizar que no se produzcan ciclos de conflictos.
- Enfoque optimista consiste en dejar hacer a las transacciones y deshacer (rollback) una de ellas si en fase de validación se descubre un conflicto.

Basado en locks

- El objetivo es usar locks para bloquear a los recursos (items) y no permitir que más de una transacción los use en forma simultanea.
- Los inserta el SGBD como instrucciones especiales.
- No es trivial el agregado de estos.
- (Lock y Unlock) tienen carácter bloqueante, cuando lo adquiero nadie mas puede adquirir un lock sobre el mismo item hasta que no se libere. (debe de ser atómico el lock - Sistemas Operativos)
- Hay locks de varios tipos, los principales son de escritura (acceso exclusivo) y de lectura (acceso compartido)
- No alcanza con usar locks por si solo.
- **No puedo adquirir un lock luego de desbloquear un lock.** (Protocolo de lock de 2 fases - 2PL) Los unlock/lock los puedo poner donde quiera mientras no haga lock despues de un unlock
- El protocolo nos dice que va a ser serializable, pero puede ocurrir un deadlock (un conjunto de transacciones quedan bloqueadas entre ellas bloqueadas a la espera de recursos que otra transacción posee.) Se pueden prevenir tomando todos los locks que se necesitan de forma preventiva, la desventaja es que no sabemos todo lo que vamos a necesitar. Se puede definir un ordenamiento de los recursos y timestamps también.
- Se utiliza el grafo de alocación de recursos para detectar deadlocks. Si encontramos un ciclo en el grafo tenemos un deadlock.
- Puede hacer inanición.
- No resuelve generalmente la lectura sucia, las otras si.
- Estructura de tipo árbol para buscar los bloques. (Se usan Arboles-B) (Están hechos en forma eficiente para almacenar en disco) Los bloques de datos (un puntero a estos) esta en las hojas de los arboles. Se aplican index locks para bloquear los recursos (bloquean nodos de un índice.) Bloqueo al padre, bloqueo al hijo, puedo desbloquear al padre. Cada nodo subsiguiente se puede bloquear si tengo el del padre. Un nodo deslockeado no puede volver a lockearse. Protocolo del cangrejo. Bloqueamos la raiz y vamos adquiriendo locks con los hijos hasta llegar a lo que queremos, liberando el padre a menos que sepamos que se puede partir el nodo hijo.

Timestamps

- A cada transacción se le asigna una marca de tiempo que se lo asigna el gestor cuando inicia la transacción.
- Los timestamps deben de ser únicos y determinaran el orden serial respecto al cual el solapamiento deberá ser equivalente.

- Se permite la ocurrencia de conflictos pero siempre que las transacciones aparezcan en el orden serial equivalente.
- No tiene deadlocks (no usa locks)
- Para cada item X se debe de mantener:
  - read_TS(X): Es el TS(T) correspondiente a la transacción mas joven que leyo el item X. (mayor TS(T))
  - write_TS(X): Es el TS(T) correspondiente a la transacción mas joven que escribio el item X. (mayor TS(T))
- La desventaja es que impone un orden 'caprichosos' porque una transacción recibió un timestamp menor a otra nos obliga el orden serial equivalente.

Reglas:

- Cuando una transacción Ti quiere ejecutar R(X), si una transacción posterior Tj modifico el item, Ti deberá ser abortada. De lo contrario se actualiza el valor de read_TS(X) y lee.
- Cuando una transacción Ti quiere ejecutar W(X), si una transacción posterior Tj leyó o escribió el item, Ti deberá ser abortada (write too late). De lo contrario se actualiza el valor de write_TS(X) y escribe.

Regla de escritura de Thomas

- Si cuando Ti intenta escribir un item que una transacción posterior Tj ya lo escribió entonces Ti puede descartar su actualización sin riesgos siempre y cuando el item no haya sido leído por ninguna transacción posterior Ti.
- Si yo soy anterior a una transacción que ya escribió y yo quiero escribirlo entonces si entre los dos timestamps nadie la leyó entonces puedo evitar esa actualización y no tengo que abortar esa transacción.

Control de concurrencia multiversion

Snapshot Isolation es una de las implementaciones posibles. El objetivo es que cada transacción vea una imagen (una 'foto') de la base de datos correspondiente al instante de inicio, Que sea una foto quiere decir todo lo que estaba commiteado solo. Me deshago de la snapshot cuando termina la ultima transacción que la estaba usando.

Cuando dos transacciones intentan modificar el mismo item gana la que commitea primero, la otra se aborta. (first-committer-wins). Por sí solo no alcanza para garantizar la serializabilidad, para eso se debe validar permanentemente con el grafo de precedencias buscando ciclos de conflictos RW y usar Locks de predicados en el proceso de detección de conflictos, para evitar la anomalía del fantasma.

Write Skew No se evita en snapshot isolation.

Para evitarlo podemos usar locks de tablas o locks de predicados (mas eficiente) (Bloquea las tuplas que podrían cumplir la condición, para eso se aprovecha la estructura de árbol y el índice que se usa). Se llaman estructuras índice, (CREATE INDEX) ayudan a evitar tener que recorrer toda la tabla. (Los creamos si nos sirve para determinado atributo)

Si tengo estas estructuras las puedo aprovechar para evitar el lock de fantasma. Bloqueamos el nodo del árbol y los bloques, por lo que nadie puede insertar o modificar lo que nos interesa. (Range lock)

Recuperabilidad

- La serializabilidad de las transacciones ya nos asegura la propiedad de aislamiento.
- Queremos que una vez que una transacción commiteo, no se deba deshacer. Ayuda a implementar durabilidad.
- **Definición**: un solapamiento es recuperable si y solo si ninguna transacción T realiza el commit hasta tanto todas las transacciones que escribieron datos antes de que T los leyera hayan commiteado. (Nunca pasa que alguien commitea cuando quienes le dieron datos todavía no commitearon)
- Hay que mirar las lecturas, si alguien lee algo que otro modifico.
- Un SGBD no debería jamás permitir la ejecución de un solapamiento que no sea recuperable.
- Si un solapamiento de transacciones es recuperable, entonces nunca será necesario deshacer transacciones que ya hayan commiteado. Aún así puede ser necesario deshacer transacciones que no aún no han commiteado. (puede que produzca rollbacks)
- Para evitar los rollbacks en cascada es necesario que una transacción no lea valores que aún no fueron commiteados. Esto es más fuerte que la condición de recuperabilidad y evita la lectura sucia. (No leo algo a menos que este commiteado)
- Los locks pueden ayudar. Protocolo de lock de dos fases estricto (S2PL) y Protocolo de lock de dos fases riguroso (R2PL). Garantizan que todo solapamiento sea serializable, recuperable y no produzca rollbacks en cascada.

Clase 15

19. Bases de datos espaciales

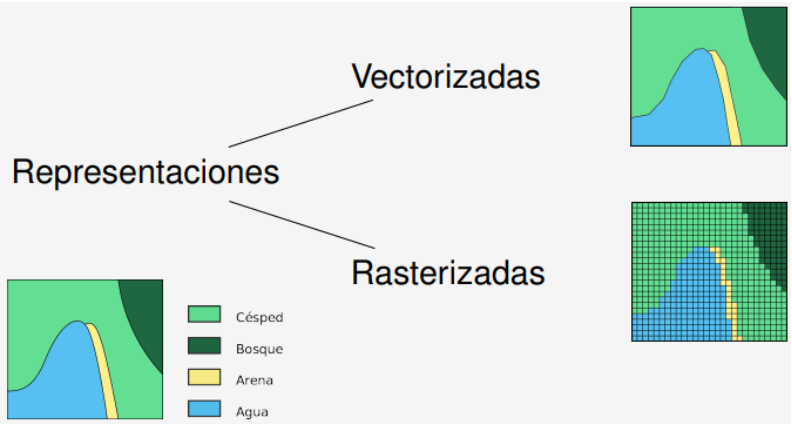
- Tenemos que ver las distorsiones que ocurren al pasar de un mapa geoide al plano. Introducción al tema de geodesia aquí
- Las bases de datos espaciales (llamadas también geográficas) permiten representar en forma eficiente objetos definidos en un espacio geométrico.
- Tienen 3 objetos simples (puntos, lineas, polígonos) y 2 complejos (objetos 3D, teselados).
- Son parte de un contexto más general, los sistemas de información geográfica GIS que permite almacenar y manipular datos geográficos y capturar, analizar y presentar visualmente datos geográficos.

Representaciones

Tenemos 2 representaciones posibles:

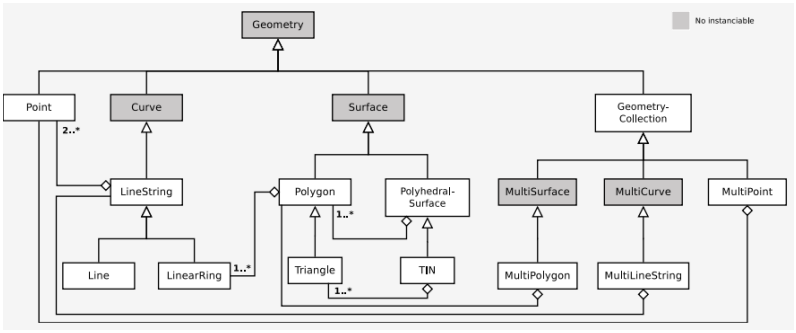
- En las representaciones vectorizadas los objetos se identifican con vectores que identifican los bordes. Tiene un alto nivel de detalle y es fácil de mantener y actualizar, pero tiene una pobre representación de datos continuos.
- En las rasterizadas los objetos se proyectan sobre una matriz de celdas. Se pueden representar datos continuos (Ej. elevación), se puede hacer un análisis cuantitativo de forma fácil, es fácil de renderizar. La desventaja es que el detalle depende de la resolución y que suelen ocupar mas espacio.

En general se trabaja con los dos tipos de imágenes juntas. (basemap) (Ej, poner un camino de GPS arriba de una imagen satelital.)



Simple Features

- Define una representación textual estándar para los objetos genéricos. Se conoce como WKT (Well Known Text). Hay otra que es Well Known Binary Representation (WKB)
- Define cómo agregar funcionalidad espacial a las bases de datos a través de una representación vectorizada, especificando con notación UML.



- Sistema de coordenadas (spatial reference system) SRID, usamos el 4326 en el taller (usado por el sistema GPS)
- spatial_ref_sys tiene los sistemas referenciales que podemos usar.
- Para buscar y combinar objetos geograficos se utilizan estructuras eficientes. Por ejemplo kd-trees o R-trees.

Taller

- Creamos base de datos, vamos a extensiones. Creamos una extensión postgis y otra postgis_raster. Luego creamos las tablas, las importamos desde el postgis.
- Nos conectamos con la base de datos e importamos los archivos poniendo el SRID en la columna. Una vez que lo hacemos seleccionamos importar.
- Hacemos el refresh de las tablas en la base de datos y ya podemos ver los datos.
- Stack Builder para instalar paquetes al postgres.
- QGis para las imágenes satelitales, instalamos el complemento y vamos a administrar base de datos donde importamos el archivo de la imagen satelital.

```
SELECT gid, nombre_est, comuna, barrio, geom
FROM primarias
WHERE gid IN (2808, 1191, 190, 559, 1512)
```

Obtener el área de todas las comunas expresada en hectáreas

```
SELECT comunas, area, ST_Area(geom::GEOGRAPHY)/10000 AS area_en_ha
FROM comunas
ORDER BY 1;
```

No esperar encontrar geometrías exactas, puede no dar lo mismo que el valor que se tiene al calcular el área.

¿Hay barrios que sean iguales a comunas? ¿Cuales?

Vemos si hay polígonos iguales.

```
SELECT c.comunas, c.geom
FROM barrios AS b, comunas AS c
WHERE st_equals(c.geom, b.geom)
ORDER BY 1 ASC
```

Hay 3 en realidad pero da 1 debido a los errores en el calculo. Si queremos los otros 2 hay que 'perdonar' la superposición.

Visualice la comuna 1 dentro del entramado de barrios

Ejemplo de unión de los barrios con la comuna.

```
SELECT st_union(c.geom, b.geom)
FROM comunas c, barrios b
WHERE c.comunas=1
```

Calcule la distancia entre todas las escuelas

```
SELECT p1.nombre_est, p2.nombre_est, ST_DistanceSphere(p1.geom,p2.geom)
FROM primarias AS p1, primarias AS p2
WHERE p1.gid < p2.gid
ORDER BY 3 DESC
```

Obtenga un ranking de las escuelas más aisladas

```
SELECT grid_1, estab_1, min(dist) AS distMin
FROM distancias AS p1
GROUP BY 1, 2
ORDER BY 3 DESC
```

Muestre las escuelas primarias y los radios censales

```
SELECT p.geom
FROM primarias AS p
UNION
SELECT c.geom
FROM censo_2010_cada AS c
```

Escuelas en comuna 1 mostrando radios censales

```
SELECT p.geom
FROM primarias AS p
WHERE p.comuna = 1
UNION
SELECT ST_union(p.geom, c.geom)
FROM primarias AS p, censo_2010_cada AS c
WHERE p.ST_Within(p.geom, c.geom) AND p.comuna = 1
```

Clase 16 y 18

En realidad se extendió mas clases el tema.

20. NoSQL

- No responden al modelo relacional.
- Surgen alrededor de los 2000 con la masificación de la Web y cambios tecnológicos.
- Necesidades de almacenar muchos datos. (Google, Amazon)
- Se tenían requerimientos
- Mayor escalabilidad para trabajar con grandes volúmenes de datos.
- Mayor performance en aplicaciones Web. Surge XML y JSON, formatos fáciles de serializar y procesar.
- Mayor flexibilidad sobre las estructuras de datos. Los SGBD relacionales son muy rígidos, agregar una columna puede ser muy costoso.
- Mayor capacidad de distribución. Viene de la mano con escalabilidad. Se busca mayor disponibilidad y tolerancia a fallas de parte del SGBD.

Los SGBD relacionales tiene limitaciones en cuanto a los joins de tablas, son costosos y el manejo de transacciones en forma distribuida no escala (Se vuelve difícil garantizar ACID cuando tenemos muchos nodos).

Lectura sugerida: Starbucks Does Not Use Two-Phase Commit

Se tenían redes cada vez mas rápidas, almacenamiento mas barato, pero velocidad de procesamiento estancada. Se pueden revisar los números del cambio aquí. Debido a esto surge la necesidad de bases de datos distribuidas.

Las bases de datos NoSQL buscan aumentar la velocidad de procesamiento y la capacidad de almacenar información. Para ello implementan un sistema de gestión de bases de datos distribuido.

Para hablar de las bases de datos no sql se necesitan algunos conceptos:

- **Fragmentación**: repartimos el contenido en muchos nodos. En uno solo no nos entra. Es la tarea de dividir un conjunto de agregados entre un conjunto de nodos. Puede ser fragmentación horizontal (los agregados se reparten entre los nodos de manera que cada nodo almacena un subconjunto de agregados. Se asigna el nodo a partir del valor de alguno de los atributos del agregado) o vertical (Distintos nodos guardan un subconjunto de atributos de cada agregado. Todos suelen compartir los atributos que conforman la clave - no es lo mas común) (En un solo nodo no nos entra, toma mucho tiempo)
- **Replicación**: si se cae uno que lo pueda manejar otro. Tiene varias ventajas, provee backup, repartir la carga de procesamiento, y garantizar la disponibilidad del sistema si se caen algunos nodos. El problema que genera es la consistencia de los datos. Cuando se usan solo como backup se dice replica secundaria. Cuando pueden hacer procesamiento también se conoce como replica primaria. La replica introduce el problema de consistencia, que un mismo item de datos tenga el mismo valor en todas las replicas. (A prueba de fallas)
- **Búsqueda (lookup)**: saber donde esta lo que necesito, que nodo tiene determinado dato. No esta este problema en una base de datos relacional. Se usan tablas de hash distribuidas.
- **Métodos de consistencia**: Que pasa si un mismo dato tiene valor distinto en distintos nodos. Puede suceder cuando los usuarios están modificando los datos. Tiene que haber una forma de controlar esto, que los usuarios se pongan de acuerdo o no pase. Surge por la replicación,
- **Métodos de acceso (Access method)**: Como llego dentro del nodo a la información que quiero de forma rápida. (LSM Trees y estructuras) diferenciales)

Clasificación

Clave-Valor

- Almacenan vectores asociativos o diccionarios, es decir pares.
- Las claves son únicas (no puede haber 2 pares con la misma clave).
- Ejemplos son Dynamo y Redis.
- Operaciones: PUT un nuevo par, UPDATE par de la clave, DELETE par de la clave, GET par de la clave
- Las ventajas que tienen son: son simples, veloces (eficiencia de acceso sobre la integridad), y escalables (se provee replicación y se pueden repartir las consultas entre nodos).
- El objetivo es consultar y guardar grandes cantidades de datos.

Dynamo

- Es el key-value de Amazon
- Esta orientada a una arquitectura orientada a servicio (SoA)
- La base de datos esta distribuida en un server cluster que posee servidores web, routers de agregación y nodos de procesamiento
- Para consistencia usa un modelo llamado consistencia eventual. Tolera pequeñas inconsistencias (valores distintos).
- De lookup usa un método de hashing consistente que reduce la cantidad de movimientos de pares necesarios cuando cambia la cantidad de nodos S. Hace que agregar nodos sea sencillo con un impacto mínimo.(no tiene nada que ver con la consistencia, solo minimiza la cantidad de movimientos)
- La función de hash consistente a partir de una clave k devuelve un valor $h(k)$ entre 0 y $2^M - 1$ en donde M representa la cantidad de bits del resultado (hashing). Este valor para un par representa en que nodo se almacena (tabla de hash distribuida)
- Al identificador de cada nodo de procesamiento (generalmente, su dirección IP) se le aplica la misma función de hash. Los nodos se van organizando virtualmente en una estructura de anillo por hash creciente. (para que si se cae alguno no reorganiza todo, si se cae alguno, el nodo siguiente se ocupa de los datos del nodo que se cayo)
- Los nodos tienen un listado de preferencia, indica cuales nodos va a replicar. Si se cae un nodo, otro nodo que no esta en este listado va a recibir los datos para mantener (el único movimiento que hay)

Orientadas a Documentos

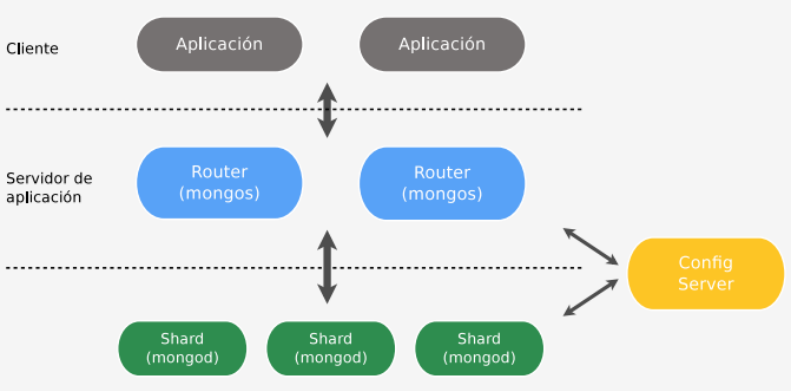
- Un documento es un agregado, la unidad estructural que contiene la información.
- No tenemos un esquema rígido (columnas).
- Un agregado es un conjunto de objetos relacionados que se agrupan en colecciones para ser tratados como unidad y ser almacenados en un mismo lugar. *Un post de FB con sus comentarios*
- Generalmente se representan con JSON, XML, YAML...

- Ejemplos son MongoDB, RavenDB...

MongoDB

- Se basa en hashes para identificar los objetos.
- No usa esquemas pero se le pueden definir.
- Documentos son JSON.
- Organiza los datos de una base de datos en colecciones que contienen documentos.
- No esta pensado para operaciones de junta, en general se pone todo junto en los documentos.
- La agregación se implementa con un pipeline, uno va definiendo las operaciones a realizar en el. La salida de uno es la entrada de otro.
- Sharding es el modelo distribuido de procesamiento. Se basa en el particionamiento horizontal de las colecciones en chunks que se distribuyen en nodos llamados Shards. La idea de estos es que estén descentralizados.

El esquema de MongoDB se puede ver como:



- Son los shards (fragmentos) que se distribuyen los chunks, los routers que reciben las consultas, y los servidores de configuración sobre los shards y routers.
- Mongo usa para particionar las colecciones una shard key. Es un atributo o conjunto de atributos.
- Es posible tener colecciones sharded y otras unsharded. Las unsharded se almacenan en un shard particular del cluster. (El shard primario)
- No es posible desfragmentar una colección ya fragmentada.
- Usar el sharding permite disminuir el tiempo de respuesta en sistemas con alta carga de consultas al distribuir el procesamiento entre varios nodos y ejecutar consultas sobre conjuntos de datos muy grandes. El objetivo es que la base de datos sea escalable.
- MongoDB nos permite que cada shard este replicado. Nos sirve para backup y responder consultas.
- El esquema de replicas es de master-slave with automated failover. Las replicas eligen entre si un master con un algoritmo distribuido. Si el master falla los slaves elijen a un nuevo master.

- Todas las operaciones de escritura se hacen sobre el master. Los slaves están como respaldo.
- Los clientes pueden especificar para mandar las consultas de lectura a los nodos secundarios.
- Durante operaciones de agregación, se puede dar el caso de que los nodos realice operaciones de forma paralela, y cuando no se pueda seguir de forma paralela (se necesita la información de los otros nodos) se le pasan al router para que lo termine la operación.

Wide Column

- Evolución de las bases de datos de clave/valor.
- Un valor particular de la clave primaria junto con todas sus columnas asociadas forma un agregado análogo a la fila de una tabla. Pero además, estas bases permiten agregar conjuntos de columnas en forma dinámica a una fila, convirtiéndola en un agregado llamado fila ancha (wide row, el nombre quedo como wide column)
- Ejemplos son Google BigTable, Apache Cassandra...

Cassandra

- No es estrictamente orientada a columnas.
- Usa esquemas.
- Tiene una arquitectura de share-nothing, no hay un estado compartido centralizado, todos los nodos son pares. Permite que sea muy escalable.
- Optimizado para ofrecer una alta tasa de escrituras.
- Cada columna es un par clave-valor asociado a una fila.
- Cada esquema puede estar distribuido en varios nodos.
- Es obligatorio definir una clave primaria.
- Cuando en una fila las columnas se repiten identificadas por el valor que toman las columnas clave, se dice que la fila se convirtió en una wide row (fila ancha).
- La clave primaria se divide entonces en clave de partición y de clustering. (La primaria permite identificar a la fila todavía)
- Toda la wide-row se almacenara contigua en disco y la clave de clustering nos determina el ordenamiento interno de las columnas.
- Restricciones sobre la clave primaria: los valores deben ser comparados por igual contra valores constantes en los predicados, si una columna que forma parte de una clustering key es usada como predicado, deben usarse las restantes columnas también.
- No existe el concepto de junta.
- No existe el concepto de integridad referencial.
- Hay que pensar de antemano que consultas se van a realizar para diseñar las tablas. Se busca que cada consulta se resuelva accediendo a una unica column family, que los resultados esten en una unica particion, respetar las reglas del uso de la clave primaria.
- Usa una estructura LSM-Tree que mantiene parte de los datos en memoria para diferir cambios sobre el indice en disco.
- Se busca acceder en forma secuencial al disco.

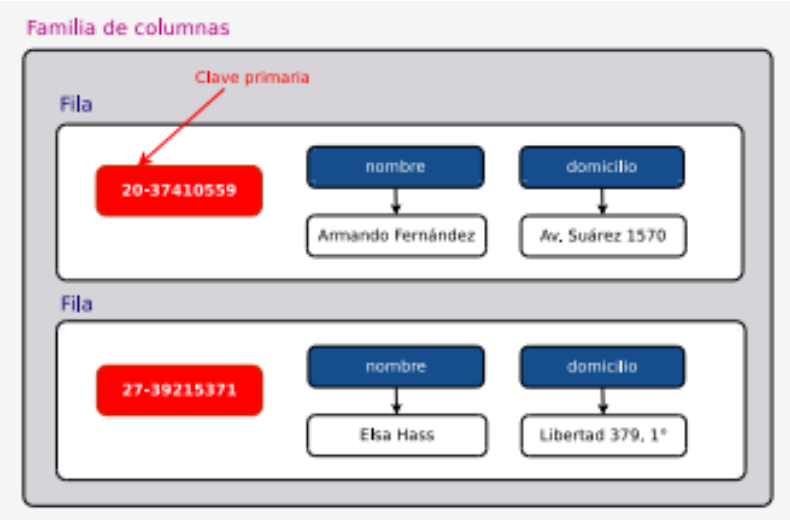


Figura 5: Esquema simple de Cassandra

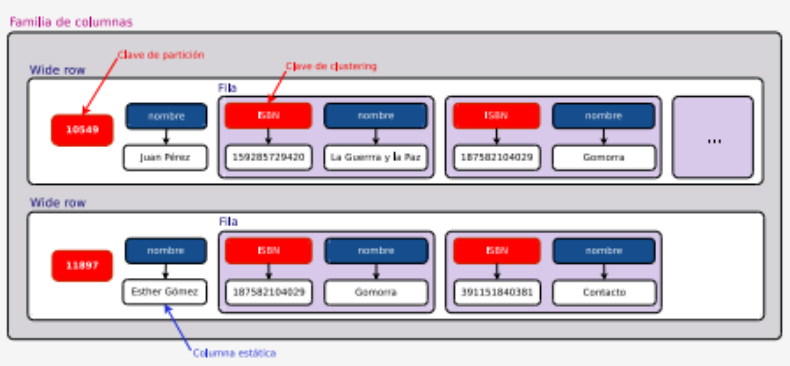


Figura 6: Esquema con Wide Row

Basadas en grafos

- Los elementos principales son nodos y arcos (ejes)
- Estas bases resultan útiles para modelar interrelaciones complejas entre las entidades.
- Mantienen una referencia a sus nodos adyacentes.
- Se almacena como una lista de adyacencias
- Sirve para encontrar un patrón de nodos conectados entre si, un camino entre nodos, la ruta mas corta, calcular medidas de centralidad asociadas a los nodos.

Neo4j

- Tiene soporte para ACID
- Formada por nodos que pueden tener distintos labels.
- Dentro de cada label el nodo tiene un conjunto de propiedades. Esta estructura no es rígida.

- Los ejes son siempre direccionales, si se quiere trabajar con no dirigidos no se crean las interrelaciones en ambos sentidos.
- Un patrón (pattern) puede especificarse a través de un nodo y sus propiedades, una interrelación y sus propiedades, o un camino y sus propiedades. A cada patrón podemos darle un nombre.
- Las funciones de agregación se realizan en el RETURN.

Consistencia

Consistencia secuencial

- Es el provisto por las bases de datos centralizadas.
- Se dice que una base de datos distribuida tiene consistencia secuencial cuando "el resultado de cualquier ejecución concurrente de los procesos es equivalente al de alguna ejecución secuencial en que las instrucciones de los procesos se ejecutan una después de otra"
- Garantizar consistencia secuencial es costoso, ya que requiere de mecanismos de sincronización fuertes que aumentan los tiempos de respuesta.

Consistencia causal

- Busca captar eventos que puedan estar causalmente relacionados
- Si un evento b fue influenciado por un evento a, la causalidad requiere que todos vean al evento a antes que al evento b.
- Dos eventos que no estén causalmente correlacionados se dicen concurrentes. No se tienen restricciones en este caso.

Consistencia eventual

- En algún momento los nodos se van a poner de acuerdo. Se basa en que son pocos los procesos que realizan modificaciones o escrituras mientras que la mayor parte solo lee.
- Deja que pase y después se vea como resolverlo, eventualmente todas las replicas son consistentes.
- Dynamo provee este sistema.
  - Se definen dos parámetros adicionales debido a que puede darse el caso de que se lea un valor desactualizado.
  - $W \leq N$: Quorum de escritura, se devuelve que la lectura fue exitosa cuando otros $W-1$ nodos confirman el valor. W=2 es el mínimo.
  - $R \leq N$: Quorum de lectura, se devuelve éxito cuando se tienen R nodos distintos. Generalmente R = 1 es suficiente. Valores mayores de R brindan tolerancia a fallas como corrupción de datos ó ataques externos, pero hacen más lenta la lectura
  - En Dynamo hay que tener en cuenta que no hay funciones de agregación, se implementan a mano o se usan herramientas que envuelven a Dynamo (Ej Spark).

Arboles

Log Structured Merge Trees LSM-trees

- Ofrece escrituras secuenciales (El B-Tree es de acceso aleatorio), es mucho más rápido que tener que posicionarse en el disco. (El B-Tree es peor para escritura, el costo de escritura es alto porque me estoy moviendo todo el tiempo en disco.)
- La desventaja es que a veces se pierde tiempo cuando los datos no están cacheados. La lectura es más torpe. (El B-Tree es mejor para lectura)
- Es usado por Cassandra, Mongo...

Map Reduce

- Es una técnica que brinda un marco flexible para el procesamiento paralelo de grandes volúmenes de datos.
- Divide la entrada en partes mas pequeñas que puedan ser ejecutadas por unidades de procesamiento para luego integrar el resultado a la salida.
- Map devuelve una secuencia de clave/valor
- Reduce recibe una clave con sus valores y devuelve valores.

Teorema CAP

Es un teorema que postula la imposibilidad de que un sistema distribuido garantice simultaneamente el maximo nivel de:

- Consistencia (Se muestre un único valor, requiere mucha sincronización)
- Disponibilidad (Cada consulta que llegue a un nodo no caido devuelva un resultado sin errores)
- Tolerancia a fallas (Que se pueda responder a consultas aun cuando algunos nodos estan caidos.)

A lo sumo podemos ofrecer 2 de las 3 garantias

En la realidad no es factible garantizar que una red no se particione, con lo cual nuestro sistema deberá necesariamente ser tolerante a particiones. Nos obliga entonces a encontrar una solución de compromiso entre consistencia y disponibilidad

BASE

- (BA) Disponibilidad básica (basic availability): El SGBD distribuido está siempre en funcionamiento, aunque eventualmente puede devolvernos un error, o un valor desactualizado.
- (S) Estado débil (soft state): No es necesario que todas los nodos réplica guarden el mismo valor de un ítem en un determinado instante. No existe entonces un "estado actual de la base de datos"
- (E) Consistencia eventual (eventual consistency): Si dejaran de producirse actualizaciones, eventualmente todos los nodos réplica alcanzarían el mismo estado.

Clase 17

21. Practica: MongoDB

- Orientada a documentos
- Sirve para información que subimos y bajamos por la Web.
- Usamos el formato JSON
- Los números son enteros o decimales.
- Es muy libre la estructura, puedo tener una colección donde cada documento sea distinto. No estamos obligados a tener esquemas.
- La desventaja es que voy a tener que realizar muchos más chequeos.
- Lo natural es tener todo junto.
- La ausencia de los datos no es algo que esta mal. A veces da mas información, por ejemplo la gente que no dice cuanto gana se sabe que generalmente es gente de ingresos altos.
- En SQL se ponen nulos para rellenar, en Mongo es con JSON igual, pero al trabajar con los datos hay que tener en cuenta las faltas. Si queremos evitar los nulos en SQL usamos nuevas tablas que se relacionen por la clave.

'Renombrando' queda:

- Una tabla es una colección ahora.
- Una fila es un documento.
- Una columna es un campo. (field)
- Un join es un documento embebido.

Sentencias CRUD

- db.<collection>.find(<query>, <projection>)
- db.user.find({firstName:"Juan"})
- El query es como el WHERE
- La proyección para decir que campos queremos ver. Con 0 no los vemos, con 1 si. Es como un SELECT.
- El id nos los trae siempre, si no lo queremos le tenemos que poner 0 si o si.
- Podemos hacer búsquedas más complejas con $regex
- Tenemos and, or, eq, gt,ge,lt, lte.
- Se le puede poner un limite de la cantidad de elementos que queremos que nos traiga.
- Varias cosas más que se pueden ver de la documentación...

Taller

Mostrar el país cuya capital es Vienna

```
db.countries.find({'capital':'Vienna'},{_id:0})
```

Mostrar los países con un área igual o menor a 20

```
db.countries.find({area: {$lte : 20}}, {_id : 0, "name.common" : 1})
```

Mostrar todos los países de América

```
db.countries.find({region: "Americas"}, {_id : 0, "name.common" : 1})
```

Mostrar todos los países de América con un área menor a 100

```
db.countries.find({region: "Americas", area: {$lt : 100}}, {_id : 0, "name.common" : 1})
```

Mostrar los países que estén en Europa o que use el euro (código: EUR) como moneda oficial

```
db.countries.find({$or:[{region:'Europe'},{currency:'EUR'}]},{_id:0,"name.common":1})
```

Mostrar las capitales de los países que no usen el dolar estadounidense (código: USD) ni el dolar canadiense (código: CAD) como moneda oficial

```
db.countries.find({$nor:[{'currency':'USD'}, {'currency':'CAD'}]},{_id:0})
```

Mostrar los países que limitan con Francia o Polonia

```
db.countries.find({'borders': {$in: ['FRA','POL']}},{'name.common':1})
```

Mostrar los países que limitan con Francia y Polonia

```
db.countries.find({\$and :[{'borders': {$in: ['FRA']}}, {'borders': {$in: ['POL']} }]},{'name.com
```

Mostrar los países cuyo nombre oficial contenga la palabra "Republic" y que tengan exactamente 3 países limítrofes

```
db.countries.find({$and: [ {"name.official": {$regex: /^Republic/} },
{ borders: {$size: 3 } } ] }, {_id:0, "name.common":1})
```

Mostrar la cantidad de libros de m´as de 400 paginas y que tengan un solo autor

```
db.books.find({$and:[{pageCount:{$gt:400}},{authors:{$size:1}}]})
```

Mostrar en forma alfabética por t´ıtulo, los libros que en su t´ıtulo o en su descripción corta contengan la palabra "web"

```
db.books.find({$or:[{title:{$regex:"web.*"}},{shortDescription:{$regex:"web.*"}}]})
        .sort({title:1})
```

Mostrar el nombre y la cantidad de paginas de 12 libros publicados, ordenados descendentemente por su cantidad de paginas

```
db.books.find({_id:0,title:1,pageCount:1})
        .sort({pageCount:-1})
        .limit(12)
```

Mostrar los libros publicados entre 2008 y 2010 (inclusive), ordenados ascendentemente por su id.

```
db.books.find({$and:[{publishedDate:{$gte:new Date("01-01-2008")}},{publishedDate:{$lt:new Date("
```

updateOne remplaza el JSON directamente, si queremos evitarlo hay que usar un parámetro adicional.

```
db.countries.deleteOne({"name.common":{$regex:"Falkland"}})
```

Clase 19

22. Practica: Agregación en MongoDB

- Los operadores van dentro del aggregate.
- $match nos permite realizar una query igual al find, es el filtrado
- $group agrupa por uno o mas atributos aplicando funciones de agregacion ($sum,$avg,$first)
- $sort para ordenar los resultados
- $limit para limitar la cantidad de resultados dados
- $sample para que me de una muestra de resultados
- $project para proyectar campos, decir si queremos que estén o no.
- $unwind abre los documentos, si un documento tiene un vector, crearía un documento nuevo identico al del vector donde en vez del vector se tiene un item (por cada uno de los que tenga). Sirve si quiero reorganizar la información.
- Accedo a los valores del documento con $, agrupo poniendo el atributo en el _id

1 Hallar el nombre de usuario y la cantidad de retweets para los tweets con más de 100000 retweets

```
db.tweets.aggregate(
[{
$match: {
    retweet_count: {
        $gt: 100000
    }
}},
{
    $project: {
        _id: 0,
        "user.name": 1,
        retweet_count: 1
    }
}])
```

2 Hallar los usuarios que hayan contestado (is reply status id != null) algún tweet, sea en español y sea entre las 12 y 13 hs. Ayuda: ISODate("2019-06-26T13:00:00.000Z")

```
 db.tweets.aggregate(
 [{
 $match: {
    lang: 'es',
    created_at: {
        $gt: ISODate('1029-06-26T12:00:00.000Z'),
        $lt: ISODate('1029-06-26T13:00:00.000Z')
    },
    in_reply_to_user_id: {
        $exists:true -----otra opcion $ne:null
    }
}},
```

```
{
$project: {
    _id:0,
    "user.name":1
    }
}])
```

3 Hallar un tweet de fecha más antigua

```
[{
$sort: {
    created_at:-1
}},
{
$limit: 1
}]
```

4 Mostrar de los 10 tweets con más retweets, su usuario y la cantidad de retweets. Ordenar la salida de forma ascendente

```
[{
$sort: {
    retweet_count:1
}},
{
$limit: 10
},
{
$project: {
    "user.name":1,
    _id:0,
    retweet_count:1
    }
}]
```

5 Encontrar los 10 hashtags más usados

```
[{
$unwind: {
    path: "$entities.hashtags",
    preserveNullAndEmptyArrays: false
}},
{
$group: {
    _id: "$entities.hashtags.text",
    cant: {
        $sum: 1
    }
}
},
{
$sort: {
    cant: -1
    }
```

```
},
{
$limit: 10
}]
```

6 Encontrar a los 5 usuarios más mencionados. (les hicieron @)

```
[{
$unwind: {
    path: "$entities.user_mentions",
    }
},
{$group: {
    _id: "$entities.user_mentions.id",
    cant: {
        $sum: 1
    }
    }
},
{$sort: {
    cant: -1
    }
},
{
$limit: 10
}]
```

7 Para los tweets que empiezan con un hashtag, mostrar su índice y el hashtag

Se puede ver con el índice también para no usar regex

```
[{
$unwind: {
    path: "$entities.hashtags",
    }
},
{$match: {
    text: {$regex: RegExp('^#')}
    }
},
{$project: {
    _id:0,
    "entities.hashtags.text":1,
    "entities.hashtags.indices":1,
    }
}]
```

8 Por cada usuario obtener una lista de ids de tweets y el largo de la misma

```
[{$group: {
  _id: "$user_id",
  ids: {$push : "$_id"},
  cant : {$sum : 1}
}}, {}]
```

9 Hallar la máxima cantidad de retweets totales que tuvo algún usuario

```
[{$group: {
  _id: {id:"$user_id",name : "$user.name"},
  retweets: {$sum : "$retweet_count"}
}},
{$sort: {
  retweets: -1
}},
{$limit: 1}]
```

10 Hallar el usuario no verificado que tuvo mayor cantidad de tweets retweeteados

```
[
    {
        '$match': {
            'user.verified': False,
            'retweet_count': {
                '$gt': 0
            }
        }
    }, {
        '$group': {
            '_id': {
                'id': '$user_id',
                'screen_name': '$user.screen_name',
                'name': '$user.name'
            },
            'retweets': {
                '$sum': 1
            }
        }
    }, {
        '$project': {
            '_id': 0,
            'id': '$_id.id',
            'screen_name': '$_id.screen_name',
            'name': '$_id.name',
            'retweets': 1
        }
    }, {
        '$sort': {
            'retweets': -1
        }
    }, {
        '$limit': 1
    }
]
```

11 Hallar para cada intervalo de una hora cuantos tweets realizó cada usuario

```
[
    {
```

```
        '$group': {
            '_id': {
                'created_at': {
                    '$hour': '$created_at'
                },
                'id': '$user_id',
                'screen_name': '$user.screen_name',
                'name': '$user.name'
            },
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$project': {
            '_id': 0,
            'id': '$_id.id',
            'screen_name': '$_id.screen_name',
            'name': '$_id.name',
            'count': 1
        }
    }
]
```

Clase 20

23. Recuperabilidad

- En la vida real se pueden producir diferentes tipos de fallas.
- Fallas del sistema por errores de software o hardware que detienen la ejecución de un programa.Ej fallas de segmentación, división por cero, fallas de memoria.
- Fallas de aplicación, provienen desde la aplicación que utiliza la base de datos. Cancelación o vuelta atras de una transacción
- Fallas de dispositivos, provienen de un daño físico
- Fallas naturales externas que provienen afuera del hardware. Caídas de tensión, terremotos, incendios

Supongamos que se produce una falla no catastrófica en el momento en que se están ejecutando muchas transacciones. La base de datos debe ser llevada al estado inmediato anterior al comienzo de la transacción. Para esto se utiliza el log que se fue armando.

El log

El gestor de recuperación de la SGBD almacena generalmente los siguientes registros:

- BEGIN T
- WRITE T, X, xold, xnew
- READ T, X
- COMMIT T
- ABORT T

Siendo T una transaccion, X un item, xold el valor viejo, xnew el valor nuevo.

El gestor se guia principalmente por dos reglas:

- WAL: Write Ahead Log, antes de escribir un item a disco, se escribe el log a disco.
- FLC: Force Log Commit, antes de realizar un commit se escribe el log a disco

Técnicas

- Actualización inmediata: Los datos se guardan en disco lo antes posible, necesariamente antes del commit de la transacción.
- Actualización diferida: Los datos se guardan en disco después del commit de la transacción.

Algoritmos de recuperación

Asumen que los solapamientos son recuperables y evitan ROLLBACKS en cascada. Si no se cumple esto hay que realizar pasos adicionales.

UNDO (Actualización inmediata)

Todo valor V_{old} asignado por una transacción que ya commiteo debe ser guardado en el log en disco antes de que su modificación por parte de otra transacción sea guardada en disco (flushed).

Los pasos del algoritmo son:

1. Cuando una transacción T_i modifica el ítem X remplazando un valor v_{old} por v, se escribe ($WRITE, T_i, X, v_{old}$) en el *log*, y se hace *flush* del *log* a disco.
2. El registro ($WRITE, T_i, X, v_{old}$) debe ser escrito en el *log* en disco (*flushed*) antes de escribir (*flush*) el nuevo valor de X en disco (WAL).
3. Todo ítem modificado debe ser guardado en disco antes de hacer *commit*.
4. Cuando T_i hace *commit*, se escribe ($COMMIT, T_i$) en el *log* y se hace *flush* del *log* a disco (FLC).

- Cuando hay un reinicio, miro el log y veo si hay transacciones no commiteadas. Si encuentro alguna la tengo que deshacer. (Se considera que la transacción commiteo cuando el registro en el log queda en disco)
- El gestor se tiene que asegurar que antes del commit el dato en memoria vaya a disco.

En el reinicio los pasos son:

1. Se recorre el *log* de adelante hacia atrás, y por cada transacción de la que no se encuentra el COMMIT se aplica cada uno de los WRITE para restaurar el valor anterior a la misma *en disco*.
2. Luego, por cada transacción de la que no se encontró el COMMIT se escribe ($ABORT, T$) en el *log* y se hace *flush* del *log* a disco.

Si vuelve a fallar se ejecuta devuelta, no cambia el resultado.

REDO (Actualización diferida)

Antes de realizar el commit todo nuevo valor v asignado por la transacción debe ser salvaguardado en el log en disco.. (Me obliga solo a guardar el log primero, no el dato.) (Se postergan los accesos a disco, las fallas no deberian de ser algo comun)

Pasos:

1. Cuando una transacción T_i modifica el item X remplazando un valor v_{old} por v, se escribe ($WRITE$, T_i, X, v) en el *log*.
2. Cuando T_i hace *commit*, se escribe ($COMMIT$, T_i) en el *log* y se hace *flush* del *log* a disco (FLC). Recién entonces se escribe el nuevo valor en disco.

Figura 7: Se usa el valor nuevo, no el viejo!

Cuando el sistema se reinicia se siguen los siguientes pasos.

1. Se analiza cuáles son las transacciones de las que está registrado el COMMIT.
2. Se recorre el *log* de atrás hacia adelante volviendo a aplicar cada uno de los WRITE de las transacciones que commitearon, para asegurar que quede actualizado el valor de cada ítem.
3. Luego, por cada transacción de la que no se encontró el COMMIT se escribe ($ABORT$, T) en el *log* y se hace *flush* del *log* a disco.

Figura 8: Se recorre de lo mas viejo a lo mas nuevo. Nos tiene que quedar lo mas nuevo en disco

- Si la transacción falla antes del commit, no se deshace nada (solo se abortan las transacciones encontradas). Si falla después de haber escrito el COMMIT en disco, hay que rehacer toda la transacción (no sabemo si estan los valores nuevos en disco).
- En el algoritmo REDO, una transacción puede commitear sin haber guardado en disco todos sus ítems modificados.
- Ante una falla previa posterior al commit, entonces, será necesario reescribir (REDO) todos los valores que la transacción había asignado a los ítems (Implica recorrer todo el log de atrás para adelante aplicando cada uno de los WRITE.)
- Si no hay checkpoints hay que revisar todo el archivo, puede haber quedado una transaccion desde el comienzo sin terminar.

UNDO/REDO

Los pasos:

1. Cuando una transacción T_i modifica el item X remplazando un valor v_{old} por v, se escribe ($WRITE$, T_i, X, v_{old}, v) en el *log*.
2. El registro ($WRITE$, T_i, X, v_{old}, v) debe ser escrito en el *log* en disco (*flushed*) antes de escribir (*flush*) el nuevo valor de X en disco.
3. Cuando T_i hace *commit*, se escribe ($COMMIT$, T_i) en el *log* y se hace *flush* del *log* a disco.
4. Los ítems modificados pueden ser guardados en disco antes o después de hacer *commit*.

Figura 9: Escribimos ambos valores, el viejo y el nuevo. Después de eso podemos mandar el valor de X a disco cuando queramos (antes o después del commit, tenemos ambos datos).

En el reinicio:

1. Se recorre el *log* de adelante hacia atrás, y por cada transacción de la que no se encuentra el COMMIT se aplica cada uno de los WRITE para restaurar el valor anterior a la misma *en disco*.
2. Luego se recorre de atrás hacia adelante volviendo a aplicar cada uno de los WRITE de las transacciones que commitearon, para asegurar que quede asignado el nuevo valor de cada ítem.
3. Finalmente, por cada transacción de la que no se encontró el COMMIT se escribe ($ABORT$, T) en el *log* y se hace *flush* del *log* a disco.

Figura 10: Deshago lo que no tiene commit, rehago lo que tiene commit

- Undo o Redo me obligan a trabajar a nivel de item, no de bloque/fila.
- Es mucho mas flexible que los otros dos.
- El log se vuelve mas grande, el doble
- Me puedo despreocupar del uso del buffer del disco con este metodo

Puntos de control

- Al reiniciar el sistema no sabemos hasta donde hay que retroceder en el archivo de log. Para esto, se utilizan checkpoints. (Indica que todo hasta ese punto ha sido almacenado en disco)
- Hay checkpoints inactivos y activos.
- Los inactivos implican que no se toman mas transacciones mientras que se esta ejecutando el copiado a disco. Se usa el registro CKPT
- Los activos toman transacciones usando dos registros para el checkpoint, BEGIN CKPT y END CKPT

Undo

Inactivo

1. Cuando se quiere hacer, se dejan de aceptar nuevas transacciones
2. Espera a que terminen todas las transacciones comenzadas, hagan su commit
3. Escribe CKPT en el log y lo vuelca a disco

Si el sistema se cae después del CKPT los items ya están guardados, durante la recuperación solo debemos deshacer las transacciones que no hayan hecho commit hasta encontrar el CKPT. Se podría borrar todo lo que esta atrás del CKPT en el log.

Activo

1. Escribimos un registro (BEGIN CKPT tactivas) con el listado de todas las transacciones activas hasta el momento. (Seguimos recibiendo transacciones)
2. Esperamos a que todas esas transacciones activas hagan su COMMIT
3. Una vez que hicieron su commit, terminamos el checkpoint. No nos importa si aparecieron mas transacciones en el medio (END CKPT)

Si ocurre un problema y queremos hacer un rollback:

- Si encontramos un END CKPT, retrocedemos hasta el BEGIN CKPT. Ninguna transaccion incompleta puede haber comenzado antes.
- Si encontramos un BEGIN CKPT solamente podemos deshacer las que están en la lista o que no terminaron.

REDO

Activo

1. Escribimos un registro BEGIN CKPT con el listado de todas las transacciones activas hasta el momento
2. Hacer el volcado a disco de todos los ítems que hayan sido modificados por transacciones que ya commitearon. (las que ya habian terminado)
3. Escribir (END CKPT) en el log y volcarlo a disco.

Si ocurre un problema y queremos hacer un rollback:

- Que encontremos primero un registro (END CKPT).En ese caso, deberemos retroceder hasta el (BEGIN , Tx) más antiguo del listado que figure en el (BEGIN CKPT) para rehacer todas las transacciones que commitearon. Escribir (ABORT, Ty) para aquellas que no hayan commiteado.
- Que encontremos primero un registro (BEGIN CKPT).Si el checkpoint llego sólo hasta este punto no nos sirve, y entonces deberemos ir a buscar un checkpoint anterior en el log

UNDO/REDO

Activo

1. Escribimos un registro con el listado de todas las transacciones activas hasta el momento
2. Volcamos a disco todos **los items** modificados antes del BEGIN CKPT
3. Escribimos END CKPT y lo llevamos al disco.

En la recuperación es posible que debamos retroceder hasta el inicio de la transacción más antigua en el listado de transacciones, para deshacerla en caso de que no haya commiteado.

Puedo rehacer cosas que estén después del BEGIN CKPT si se llego al END. Lo de antes de llevo a disco.

Clase 21

24. Seguridad

Seguridad de la información: Es el conjunto de procedimientos y medidas para proteger a los componentes de los sistemas de información

Implica:

- **Confidencialidad**: La información no sea ofrecida a individuos que no están autorizados a verla.
- **Integridad**: Asegurar la correctitud durante el ciclo de vida de la información. Por ejemplo alterar las notas
- **Disponibilidad**: Asegurar que la información este disponible cuando las personas autorizadas la requieran.
- **No repudio**: Alguien que accedió a la información no pueda negar haberlo hecho. Que quede un registro del acceso.

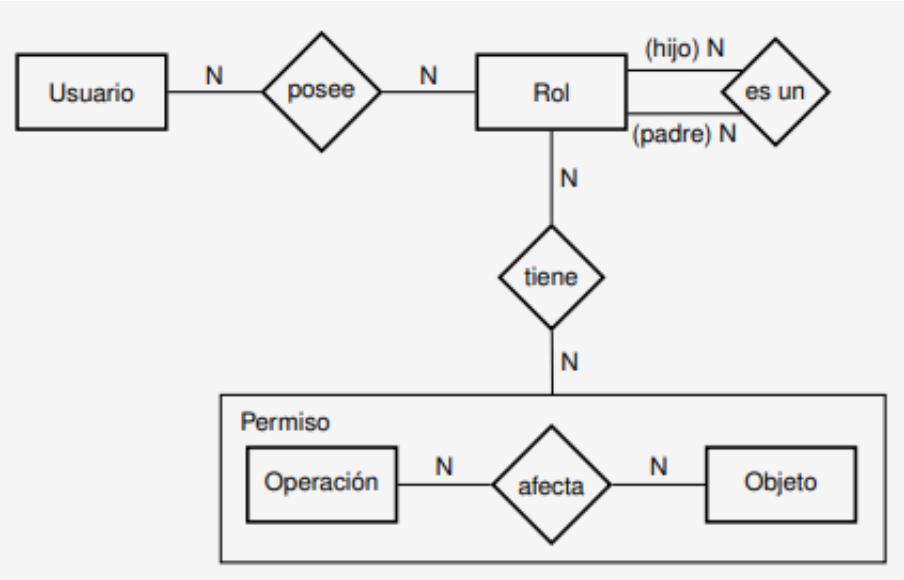
Hay que cubrir la información en distintos niveles (Personas, aplicaciones, red, OS, SGBD, archivos). Ahora nos focalizamos en lo que puede hacer el SGBD.

Control de acceso basado en roles (RBAC)

Esta basado en definir roles para las distintas actividades y funciones desarrolladas por los miembros de una organización, con el objetivo de regular el acceso de los usuarios a los recursos disponibles

Los elementos/entidades son:

- Usuarios: son las personas
- Roles: son conjuntos de funciones y responsabilidades
- Objetos: son aquello a ser protegido
- Operaciones: son las acciones que pueden realizarse sobre objetos
- Permisos: Acciones concedidas o revocadas a un usuario o rol sobre un objeto determinado.

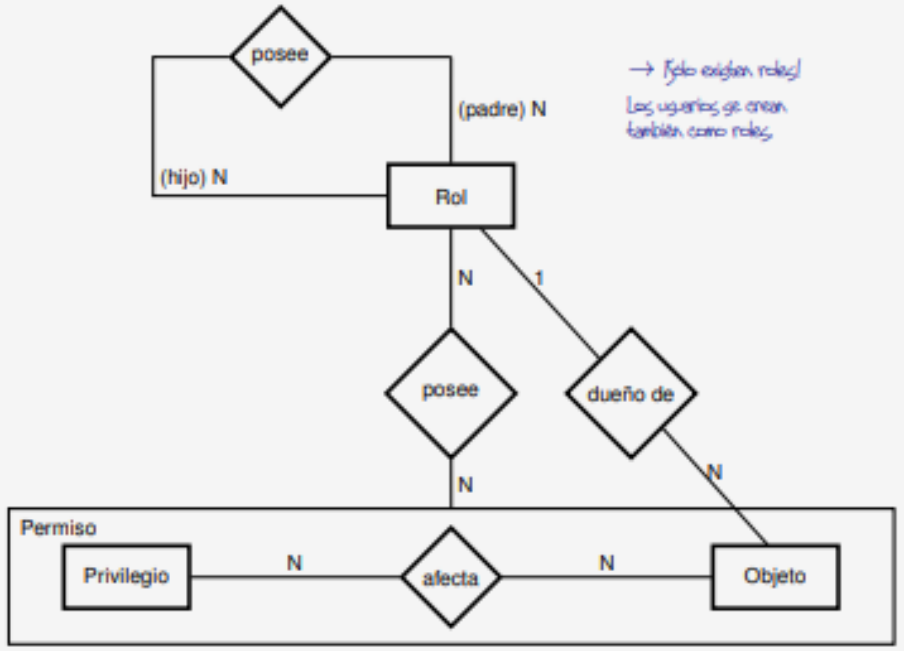


Soporta tres principios:

- Criterio del menor privilegio posible, si un usuario no va a realizar una operación, no debería de tener los permisos para realizarla.
- División de responsabilidades, nadie debe de tener suficiente privilegios para usar el sistema en beneficio propio. Necesita de roles excluyentes. *Por ejemplo, Que el pago de una factura a un proveedor requiera la aprobación de un empleado de Pagos y la carga de los datos de los proveedores deba realizarla un empleado de Compras.*
- Abstracción de datos: los permisos son abstractos, dependen del objeto en cuestión.

Autenticación y permisos en SQL

En Postgres el esquema queda de la siguiente forma:



- Postgres permite definir políticas a nivel de fila. Solo puede haber algunas filas según mis permisos al hacer un SELECT.
- Solo veo hasta donde tengo permisos al ver la tabla.
- Solo el dueño puede crear o borrar la tabla. Se puede cambiar el dueño. Lo mismo pasa con las databases y los esquemas. DROP y ALTER no son privilegios que pueden darse.
- El dueño de un objeto tiene todos los privilegios sobre el objeto, y puede extenderlos a los usuarios que desee.
- Los superusuarios tienen todos los privilegios sobre todos los objetos, y son irrevocables.

SQL Injection

Son una de las falla de seguridad más frecuentes en sistemas de bases de datos relacionales. Se manipula a nivel aplicación las variables de entrada de una consulta SQL. Sus consecuencias pueden ser graves.

La solución es hacer todo lo que se pueda para evitarlo.

- Una solución es usar una función de escape sobre los parámetros.
- Otra es aprovechar los mecanismos de control de acceso. Desde afuera se conectan con algún usuario especifico al que controlamos los permisos.
- Validar los parámetros, castear cada dato al tipo correspondiente antes de anexarlo a la consulta.
- Usar consultas parametrizadas.

Clase 22

25. Practica: Neo4J

- Orientada a grafos
- Forma distinta de modelar.
- Aplica a una gran cantidad de problemas, relaciones, distancias
- Se tienen 2 elementos, nodos y arcos
- Las ventajas son
  - Que se pueden ver patrones de nodos conectados entre si
  - Encontrar caminos entre nodos
  - La ruta mas corta
  - Medidas asociadas al grafo (centralidad,proximidad,periferia)

Arcos

- Tienen un único tipo
- Pueden tener propiedades
- Tienen una dirección si o si
- Vinculan 2 nodos o con si mismo

Consultas

```
MATCH patron
[WHERE filtros]
RETURN respuestas
[ORDER BY expresiones]
[SKIP cant LIMIT cant]
```

- patrón:estructura que quiero
- filtros, como sql

Patrón de nodos

- (alias:etiqueta:filtros)
- Es case-sensitive
- Manejo de labels: No tengo que definir esquemas

Taller

1 Muestre en orden alfabetico, las 10 primeras pelıculas del genero (genre) Science Fiction

```
MATCH (n:Movie)
WHERE n.genre = "Science Fiction"
RETURN n
ORDER BY n.title
LIMIT 10
```

2 Muestre 20 peliculas del genero "Drama" y cuyo estudio contenga la palabra "Pictures", devolviendo el titulo de la pelicula y el nombre del estudio

```
MATCH (n:Movie)
WHERE n.genre = "Drama" AND n.studio=~".*Pictures.*"
RETURN n.title, n.studio
LIMIT 20
```

Otra opción es con n.studio CONTAINS "Picture"

3 Muestre el nombre y fecha de nacimiento de 20 personas que sean actores Y directores, y que tengan fecha de nacimiento registrada

```
MATCH (n:Actor:Director)
WHERE n.birthdate IS NOT NULL
RETURN n.name, n.birthdate
LIMIT 20
```

4 Muestre el grafo de las películas filmadas por Oliver Stone

```
MATCH (os:Director{name: 'Oliver Stone'})-[d:DIRECTED]-(pelis:Movie)
RETURN os, pelis LIMIT 20
```

```
MATCH (p:Director{name=''Oliver Stone''}) - [d:DIRECTED] -> (m:Movie)
RETURN p,d,m
```

5 Muestre el grafo de todos los co-actores de Tom Hanks

```
MATCH (th:Actor{name=''Tom Hanks''}) -[a: ACTS_IN]-> (m: Movie),
  (m) <- [a2: ACTS_IN]- (ca:Actor)
RETURN  th, a, a2, ca
```

6 Muestre 10 actores que están a distancia 4 de Kevin Bacon

```
MATCH (n:Actor)-[*4]-(kb:Actor{name:'Kevin Bacon'})
RETURN n
LIMIT 10
```

Alias de subgrafos

- Para usar funciones sobre los subgrafos
  - Length
  - Relationships
  - Nodes
- Se pueden usar como booleanos
- Se pueden poner varios subgrafos en el MATCH
- Se pueden tener OPTIONAL MATCH que es similar a un outer join
- Caminos mas cortos, son funciones que reciben un subgrafo: shortestPath, allShortestPaths

7 Muestre las películas que Clint Eastwood dirigió y en las que no actuó

```
MATCH (d:Actor:Director{name:'Clint Eastwood'})-[:DIRECTED]->(m:Movie)
WHERE NOT (d)-[:ACTS_IN]->(m)
RETURN d,m
```

8 Muestre un camino más corto de Meg Ryan a Kevin Bacon.

```
MATCH r = shortestPath( (n{name:'Meg Ryan'})-[*]-(m{name:'Kevin Bacon'})  )
RETURN r
```

9 Muestre los actores que trabajaron en la trilogía completa de The Matrix ("The Matrix", "The Matrix Reloadedz "The Matrix Revolutions")

```
MATCH (a:Actor)-[:ACTS_IN]-(m:Movie{title:'The Matrix'}),
      (a)-[:ACTS_IN]-(m2:Movie{title:'The Matrix Reloaded'}),
      (a)-[:ACTS_IN]-(m3:Movie{title:'The Matrix Revolutions'})
RETURN a
```

Otra opción

```
MATCH (a:Actor)-[:ACTS_IN]-(m:Movie{title:"The Matrix"})
WHERE (:Movie{title:"The Matrix Revolutions"})<-[:ACTS_IN]-(a)-[:ACTS_IN]->(:Movie{title:"The Mat
RETURN a
```

Funciones de agregación

- Ya incluirlas en el RETURN nos devuelve lo que pedimos
- Si devuelvo algo sin una función de agregación, se agrupa por esta propiedad.
- WITH reemplaza al return (Seria como el HAVING en sql)
- Permite manipular resultados antes de pasar a la siguiente parte de la consulta
- Se puede usar para filtrar agregación
- Con WITH tengo que definir alias a lo que voy a estar trabajando con AS.

10 Obtenga el nombre de los 10 directores que han trabajado con más actores (y la cantidad de actores)

```
MATCH (d:Director)-[:DIRECTED]-(m:Movie), (a:Actor)-[:ACTS_IN]-(m)
RETURN d.name, COUNT(DISTINCT a) AS cantidad
ORDER BY cantidad DESC
LIMIT 10
```

11 ¿Quien/es es/son el/los actor/es más joven/es de la base de datos?

```
MATCH (a:Actor)
WITH MAX(a.birthday) AS mas_joven
MATCH (j:Actor)
WHERE j.birthday=mas_joven
RETURN j
```

12 Devuelva el nombre de personas que hayan actuado en al menos 10 películas y hayan dirigido al menos 5

```
MATCH (person:Person)-[acts:ACTS_IN]->(:Movie)
WITH COUNT(acts) as cant_actuaciones, person
MATCH (person:Person)-[directs:DIRECTED]->(:Movie)
WITH COUNT(directs) as cant_direcciones, cant_actuaciones, person
WHERE cant_direcciones >= 5 AND cant_actuaciones >= 10
RETURN person.name, cant_direcciones, cant_actuaciones
```

13 ¿A qué distancia se encuentra el actor más viejo de Kevin Bacon?

```
MATCH (a:Actor) WITH MIN(a.birthday) AS mas_viejo
MATCH (j:Actor) WHERE j.birthday=mas_viejo
MATH g=shortestPath((j)-[*]-(k:Actor{name:'Kevin Bacon'})
RETURN length( g )
```

ABM

- No es lo mismo crear nodos que arcos
- Para nodos es CREATE y el patron del nodo
- Para arcos es con MATCH para vincular los nodos y un CREATE para crear el arco, por cada arco

Clase 23

26. Procesamiento y Optimización de consultas

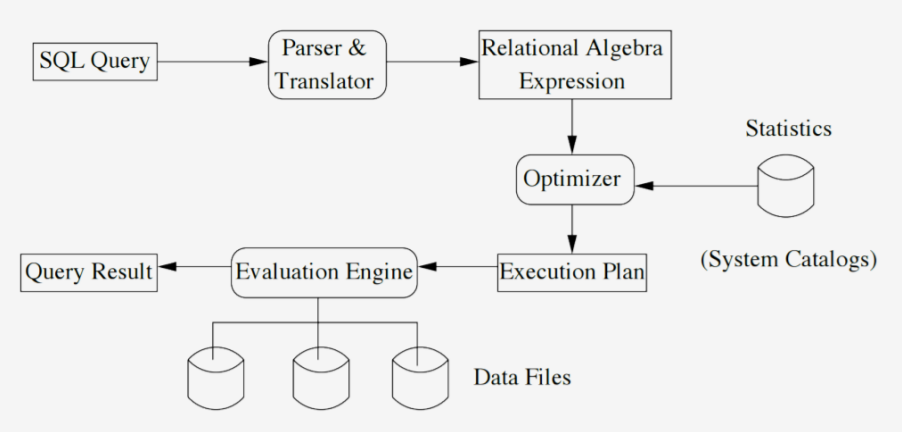


Figura 11: Representación ideal

- Parser y traductor: la parsea a la consulta, se fija que sea sintácticamente correcta y compile, la traduce a una expresión de álgebra relacional
- El optimizador analiza la expresión para llegar a un plan de ejecución: Determina con que métodos de acceso y algoritmos ejecuta lo determinado. Utiliza estadísticas para analizar esto.
- Después queda solo evaluar para obtener el resultado de la consulta.

Información de catalogo

Es utilizada para estimar costos y optimizar las consultas.

- n(R): Cantidad de filas que tiene una relación R.
- B(R): Cantidad de bloques en disco que ocupa la tabla R
- V(A,R): Cuantos valores distintos toma el atributo A en la relación R. La variabilidad no la mantiene siempre actualizada, n y B si.
- F(R): Cantidad de tuplas de R que entran en un bloque (factor de bloque) F(R) = n / B . (Esta también 1/F(R) : Que fracción de un bloque ocupa una tupla.)

El SGBD corre un proceso que realiza un análisis de estas estadísticas. También almacena información de los índices. En este caso que altura y que largo tienen.

- Heigth(I(A,R)): Altura del índice de búsqueda I por el atributo A de la relación R.
- Length(I(A,R)): Cantidad de bloques que ocupan las hojas del índice I.

Mantener actualizadas las estadísticas en cada operación ABM puede ser costoso, por lo que se hacen con cierta periodicidad.

- La expresión se optimiza a través de una heurística y utilizando reglas de equivalencia, obteniendo un plan de consulta.

- Cada plan de consulta lógico se materializa para obtener un plan de ejecución en el que se indica el procedimiento físico. Estructuras de datos a usar, índices, algoritmos, etc
- Para comparar distintos planes de ejecución necesitamos estimar el costo. Hay varios costos posibles que se pueden tomar, pero el que mas afecta en las bases de datos relacionales centralizadas termina siendo el del acceso a disco.

Índices

- Los índices son estructuras de búsqueda almacenadas y actualizadas por el SGBD que agilizan la búsqueda de registros a partir del valor de un atributo o conjunto de atributos.
- Puede implementarse con distintas estructuras de datos. (Arboles, tablas de hash)
- Se clasifican en 3 tipos:
  - Primarios, el índice es sobre el campo de ordenamiento clave de un archivo ordenado de registros.
  - De clustering, un índice por atributo no clave que ordena al archivo
  - Los secundarios, son sobre campos que no son de ordenamiento del archivo.
  - Solo se puede tener un único índice primario o de clustering.
- No se tiene un SQL estándar para la creación de los índices, pero en general es CREATE [UNIQUE] INDEX *nombreIndice* ON *tabla*

Costos

Selección

- Partimos de una selección básica (condición simple)
- Las estrategias que se tienen son:
- **File scan**: se recorre o escanea el archivo buscando aquellos que cumplan la condición. El costo es B(R) (Se recorrieron todos los bloques)
- **Index scan**: usan un índice para la búsqueda.
  - Búsqueda con **índice primario**: solamente una tupla puede satisfacer la condicion, por lo que el costo va a ser (si se usa un arbol) $Height(I(A,R))+1$ o (si se usa un hash) 1
  - Búsqueda con **índice de clustering**: Cuando Ai no es clave pero se tiene un índice de ordenamiento por el (clustering). Las tuplas están contiguas en los bloques, los cuales estarán disjuntos. El costo resulta entonces $Height(I(A,R)) + \lceil \frac{n(R)}{V(A_i,R)\cdot F(R)} \rceil = Height(I(A,R)) + \lceil \frac{B(R)}{V(A_i,R)} \rceil$
  - Búsqueda con **índice secundario**: cuando Ai no tiene un índice de clustering pero existe un índice secundario asociado a el. El costo es $Height(I(A,R)) + \lceil \frac{n(R)}{V(A_i,R)} \rceil$
- Estos cálculos pueden extenderse a otras formas de comparaciones. (Mayor, menor, distinto, etc)
- Si la selección involucra la conjunción de varias condiciones simples, pueden adoptarse distintas estrategias.
- Si uno de los atributos tiene un índice asociado se aplica primero esta condición y luego se selecciona del resultado aquellas tuplas que cumplen las demás condiciones. (El costo es solo el índice)

- Si hay un índice compuesto que tiene a varios atributos se utiliza este índice y se selecciona las que cumplan.
- Si hay índices simples para varios atributos se pueden utilizar los índices y después interesar los resultados.
- Si la selección tiene una disyunción de condiciones simples se aplican las mismas por separado y luego se unen los resultados. Si no tiene un índice alguna se hace por fuerza bruta.

Proyección

- X es el atributo de la proyección. (X' para referirnos con los duplicados)
  - Si X es superclave no hay que eliminar duplicados, por lo que el costo es B(columna)
- Si X no es superclave, hay que eliminar duplicados. Para esto se puede ordenar la tabla o usar una estructura de hash. Eliminar duplicados depende mucho de la memoria
  - Para ordenar, depende de la cantidad de memoria que tengamos. Si tenemos $B(\pi_{X'}(R)) \leq Mem$ se puede hacer directo en memoria, solo tenemos de costo leerlo todo. Si no nos alcanza la memoria, hay que hacer un ordenamiento externo (en disco) cuyo costo es: $2 \cdot B(R) \cdot \lceil log_{Mem-1}(B(R)) \rceil - B(R)$ El logaritmo representa la cantidad de etapas del sort. (Cargamos una cantidad de bloques a memoria, los ordenamos y los guardamos en otro lugar en disco. Cargo los siguientes bloques, ordeno y guardo. Así sucesivamente hasta recorrer todo. Hecho esto se hace una etapa de merge, me fijo cual es el mas chico de todos los primeros de los recorridos y avanzo en esa partición cargando el siguiente. Se hace esto hasta recorrer todo y que nos quede el archivo ordenado.) La formula cuenta cuantas veces moví cada dato de disco a memoria. (La resta es sin el almacenamiento final en disco, si lo quiero contar se saca)
  - Usando un hash, si entra en memoria se puede hacer el hashing en memoria directo. Si no, se usa el hashing externo (en disco), se hacen particiones en disco y se eliminan duplicados ahí. Dos elementos van a la misma partición. El costo es $B(R)+2B(\pi_{X'}(R))$ (Aplicamos la función de hash a cada bloque, al resultado cuando se llene el bloque con un valor lo movemos devuelta a disco dependiendo de lo que de. Cuando se termino cargamos a memoria estos bloques y tomamos los valores unicos (deberian de estar en el mismo bloque)) (En memoria tenemos una cantidad de bloques mas uno para traer el bloque de disco y aplicar la función de hash a las filas.)
- Si la consulta SQL no tiene DISTINCT el resultado va a ser siempre B(R)

Operaciones de conjuntos

- **Unión**: Se hace una tabla temporal que tenga los datos ordenados (en memoria o externo) y sin duplicados de cada parte de la unión y después se unen ambas tablas (merge) de datos ordenados. (Algoritmo dado en Algoritmos 1) Vienen ordenados por como lo hace el gestor, no es algo que le dijimos que haga. Como el estándar no obliga la forma de salida, lo da ordenado. El costo es $cost(Ordenar_M(R)) + cost(Ordenar_M(S)) + 2B(R) + 2B(S)$
- **Intersección**: La intersección es similar, si hay cosas iguales, avanzo ambas y devuelvo una. Si hay destinos avanzo la menor.
- **Diferencia**: Se devuelven las que están en R pero no en S. Se aplica el algoritmo.

Junta

Existen distintos métodos para calcularla.

- **Loops anidados**: Lo que hace es probar todos los bloques contra todos los bloques. Carga a memoria 2 bloques (uno de cada lado) y compara todos contra todos. Una vez hecho eso, saca uno (el de la derecha), carga otro y vuelve a comparar. (Así hasta que pase por toda la relación de la derecha). Nos conviene usar como pivote el mas chico. El hecho de comparar todos con todos es lo que hace la junta muy costosa, escala mal. El costo es de $min(B(R)+B(R)B(S), B(S)+B(R)B(S))$ en el peor caso, en el mejor caso se cargan ambas relaciones a memoria y ese es el único costo.
- **Único loop**: Vamos a tener un índice para acceder a los datos. Si alguna tabla tiene un índice para el atributo de junta, voy a poder recorrer a través de el. Se agarra la primera fila de S, voy al indice de R y encuentro el puntero al bloque. Junto las filas. Hay que recorrer cada fila de S. Desaparece el termino cuadrático. La junta por índice no mejora porque tenga mas memoria. El costo si el indice es primario es de $B(S) + n(S)(Height(I(A,R)) + 1)$, si es de clustering $B(S) + n(S)(Height(I(A,R)) + \lceil \frac{B(R)}{V(A_i,R)} \rceil)$, y si es secundario $B(S) + n(S)(Height(I(A,R)) + \lceil \frac{n(R)}{V(A_i,R)} \rceil)$
- **Sort-merge**: Consiste en ordenar los archivos de cada tabla por el/los atributos de junta, si entra en memoria el costo es solo cargarlo, si no, se usa un algoritmo de sort externo. Se ordena R y se vuelve a guardar en disco. El costo es $B(R) + 2B(R) \cdot \lceil log_{Mem-1}(B(R)) \rceil + B(S) + 2B(S) \cdot \lceil log_{Mem-1}(B(S)) \rceil$
- **Método de junta hash**: La idea es particionar las tablas R y S en m grupos uysando una funcion de hash aplicada sobre los atributos de junta X. Si m fue escogido de manera que para cada par de grupos (Ri, Si) al menos uno entre en memoria y sobre un bloque de memoria para hacer desfilar al otro grupo, el costo total es de $3(B(R) + B(S))$

Pipelining

Sucede cuando el resultado de un operador puede ser procesado por el operador siguiente en forma parcial sin la necesidad de esperar a que haya terminado. Se suele usar siempre que sea posible.

Al calcular el costo de dos operadores anidados $O_2(O_1(R))$ debemos considerar que en caso de utilizar pipelining no será necesario tener todos los bloques de la salida de O_1 para comenzar a calcular O_2. En particular, no tendremos que materializar toda la salida de O_1 por falta de espacio en memoria.

Estimación de la cardinalidad

- Queremos estimar el tamaño que tiene algún resultado intermedio. Debe de ser precisa, fácil de calcular y no depender de la forma en que se calculo la relación intermedia.
- Para la **selección** se usa la variabilidad. La estimación es de $\frac{n(R)}{V(A_i),R}$. (Se llama selectividad de A_1 en R a $\frac{1}{V(A_i),R}$) Otra opción es usar un histograma, es útil para valores concretos.
- Para la **junta** la estimación es de $\frac{n(R) \cdot n(S)}{max(V(R,B),V(S,B))}$.

Reglas de equivalencia

- Selección
  - $\sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_n}(R) = \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$ (Cascada)
  - $\sigma_{c_1 \vee c_2 \vee \ldots \vee c_n}(R) = \sigma_{c_1}(R) \cup \sigma_{c_2}(R) \cup \ldots \cup \sigma_{c_n}(R)$
  - $\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_2}(\sigma_{c_1}(R))$ (Conmutatividad)
- Proyección
  - $\pi_{X_1}(\pi_{X_2}(\ldots(\pi_{X_n}(R))\ldots)) = \pi_{X_1}(R)$ (Cascada)
  - $\pi_X(\sigma_{cond}(R)) = \sigma_{cond}(\pi_X(R))$ (Conmutatividad con σ)
- Producto cartesiano y junta
  - $R \times S = S \times R$ (Conmutatividad)
  - $R * S = S * R$
  - $(R \times S) \times T = R \times (S \times T)$ (Asociatividad)
  - $(R * S) * T = R * (S * T)$

- Operaciones de conjuntos
  - $R \cup S = S \cup R$ (Conmutatividad)
  - $R \cap S = S \cap R$
  - $(R \cup S) \cup T = R \cup (S \cup T)$ (Asociatividad)
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- Otras mixtas
  - Dado $\sigma_c(R * S)$, si c puede escribirse como $c_R \wedge c_S$, con c_R y c_S involucrando sólo atributos de R y de S respectivamente, entonces:

    $$\sigma_c(R * S) = \sigma_{c_R}(R) * \sigma_{c_S}(S)$$

    (Distribución de la selección en la junta)
  - Dado $\pi_X(R * S)$, si todos los atributos de junta están incluídos en X, entonces llamando X_R y X_S a los atributos de R y S que están en X respectivamente:

    $$\pi_X(R * S) = \pi_{X_R}(R) * \pi_{X_S}(S)$$

    (Distribución de la proyección en la junta)

Heurísticas de optimización

- Se realizan las selecciones lo mas temprano posible
- Productos cartesianos por juntas
- Proyectar para descartar atributos no usados lo antes posible (entre selección y proyección priorizar la selección)
- Si hay varias juntas realizar la mas restrictiva primero

Clase 24

27. Practica: Costos

- Medimos en unidades de I/O a disco
- Cualquier otro proceso que se haga en comparación tiene un costo ínfimo.
- Cuando tenemos varias tablas en un join, empezamos por la que nos va a dejar menos resultados.
- Tabla mas pequeña es la que usamos en el ciclo externo (algoritmo para la junta)
- Es para disminuir el costo
- Analizar arboles que estén ramificados a izquierda o derecha (left-deep o right-deep) para acotar las posibilidades.
- **R-trees**: Indexan objetos geométricos a través de su minimum boungind rectangle (MBR) Pongo en las hojas del R-tree los MBR de las partes que contienen cada hoja.
- **Quad-trees**: El espacio se descompone en cuadrantes disjuntos. Cada cuadrante lo subdividimos si tiene un mínimo de puntos. Se usa para indexar puntos geométricos o datos rater en dos dimensiones.

Índices

- Su objetivo es acceder de forma mas rápida a la búsqueda de datos. También es hacer cumplir las reglas de negocio (Ejemplo no hay padrones duplicados. La estructura sirve para hacerla cumplir)
- Cluster: El orden del índice con el orden de los datos coinciden. El orden en que vienen los datos del índice es como vienen los datos en el archivo
- No cluster: Es cuando no coinciden los ordenes, se cruza del índice a los datos con el camino que toma otra entrada del índice.
- La misma consulta se comporta de forma distinta dependiendo del tipo de índice.
- Simples: Están creados sobre un único atributo. I(A), I(B)
- Compuestos: Están creados por 2 o mas atributos. I(A,B) != I(B,A). El orden de los atributos dentro del índice es importante para la optimización de la consulta. Conviene poner en el primer lugar el atributo mas discriminante.
- Pueden tener duplicados o no, con duplicados seria que nos manda a un lugar donde hay varias entradas. (Ejemplo índice por comuna, una comuna nos manda a una entra con muchos alumnos.)
- Podemos indexar parcialmente las tablas. (De forma completa o densa significa que cada registro de la tabla tiene su representación en el índice)
- El índice sirve al momento de acceder a la estructura física, si proyectamos lo perdimos, ya no nos sirve.

Clase 25

28. Data warehousing

Se tienen datos estaticos y dinamicos en las empresas, el volumen de datos se relaciona con el tamaño de la organizacion. Debe de ser previsto para dimensionar la base de datos correctamente. Esta capacidad se conoce capacidad transaccional. (Dar a basto para procesar los datos)

OLTP

- On-line transaction processing
- Datos que se generan dinámicamente
- Capacidad transaccional es la capacidad para procesar el volumen de datos

Arquitectura de 3 capas

- Presentación, la interfaz en la que el usuario carga sus datos.
- Lógica, es la capa intermedia, hace de servidor. Recibe la consulta y la ejecuta.
- Capa de datos, los nodos de almacenamiento.

Olap

- On-line analitical processing
- Se empezó a hacer relevante extraer datos de información ya almacenada.
- Se tiene registro de los datos y la capacidad de procesarlos, surgió la idea de aprovecharlos para tomar decisiones.
- Para esto hace falta reducir la cantidad de datos y poder expresar consultas mas complejas
- Codd propone 12 reglas, se muestran 6:
  - Vista conceptual multidimensional, mantener los datos en una matriz, cada dimension representa un atributo.
  - Manipulación intuitiva de datos, se debe de poder diseñar la vista conceptual con una interfaz amigable.
  - Accesibilidad, combinar datos que vienen de distintos lugares (mediarlos)
  - Extracción batch e interpretativa, se debe de poder almacenar el resultado del procesamiento batch. Debe de poder mostrarse también.
  - Modelos de análisis, poder responder consultas de tipo estadístico o predicativo.
  - Arquitectura cliente-servidor, debo de poder conectarme y ver el data warehouse

Las aplicaciones OLAP generalmente se ejecutan con una copia paralela de la base de datos que se conoce como data warehouse (integran datos provenientes de fuentes de datos heterogéneas).

Modelado conceptual

- Nuestro objetivo es definir una serie de medidas numéricas sobre un conjunto de atributos a los que denominaremos dimensiones
- Debemos definir cuales serán las dimensiones del data warehouse
- A la medida numérica asociada a un valor concreto de cada una de las dimensiones la llamamos hecho.
- El diagrama de estrella permite comunicar la estructura de hechos y dimensiones de un data warehouse.

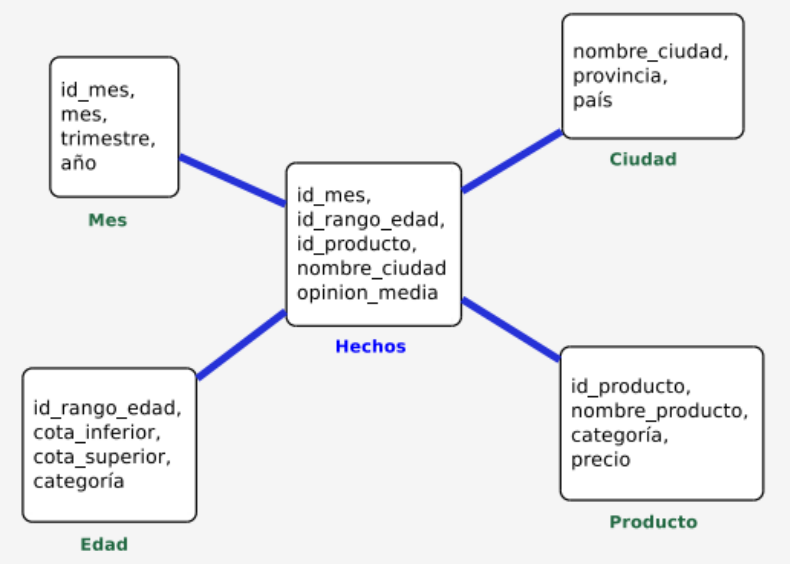


Figura 12: Ejemplo diagrama estrella

Modelo lógico

- La tabla de hechos guardará información sumarizada, de acuerdo a las dimensiones que nos interesará explorar.
- La forma de almacenamiento depende de las implementaciones. Ej MOLAP, ROLAP, HOLAP

Operaciones

- Operación roll-up consiste en agregar los datos de una dimensión subiendo un nivel en su jerarquía. Por ejemplo, si tenemos el total de ventas por ciudad y producto, podríamos hacer un roll-up de la ciudad para obtener un total por provincia y producto.
- Operación drill-down es la contraria a roll-up.
- La operación de pivoteo consiste en producir una tabla agregada por un subconjunto del conjunto de dimensiones en cierto orden deseado.
- La operación de slicing y dicing permiten realizar una selección en una dimensión o mas.

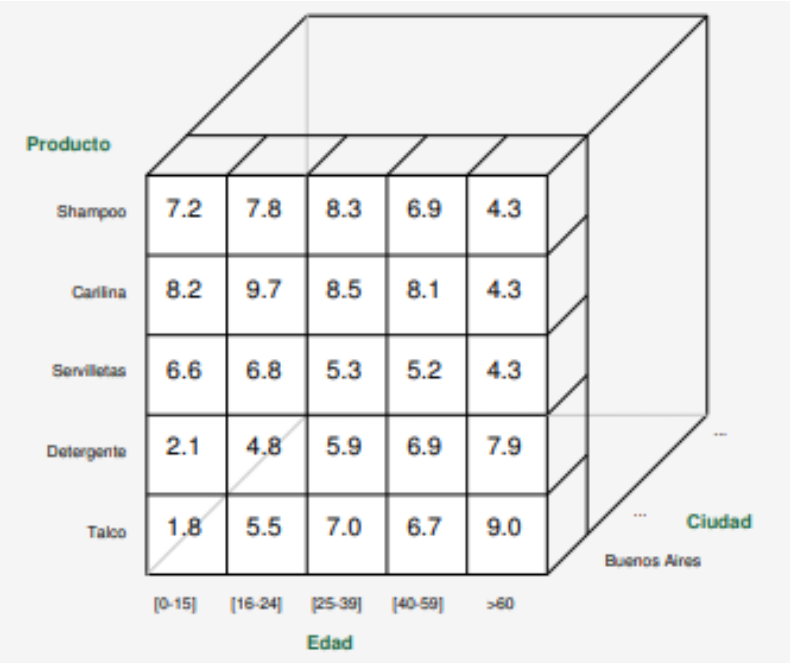


Figura 13: Ejemplo MOLAP

Clase 26

29. Practica: Resolución de consultas

- Una consulta SQL se transforma en un árbol de consultas. Buscamos ver que árbol utilizar.
- Para calcular los costos nos interesa conocer los meta datos
  - n: cuantas filas tiene
  - B: cuantos bloques tiene
  - F: factor de bloque, cuantas filas entran en un bloque
  - V: variabilidad de un atributo en una tabla (valores distintos)
  - Height: altura del índice
- Buscamos realizar las selecciones lo antes posible, reemplazar productos cartesianos y selecciones por juntas, proyectar atributos lo antes posible priorizando la seleccion y realizar las juntas más restrictivas primero.
- El objetivo es reducir el tamaño de relaciones intermedias

Selección

- File Scan: Costo B(R), se recorre toda la tabla.
- Index Scan: height(idx) + variante dependiendo del índice

Siempre acordarse de redondear hacia un valor entero.

Junta

- Loops anidados: La cantidad de memoria ayuda mucho en los loops anidados
- Único loop: Es cuando tenemos un índice que podemos aprovechar. Arranca por el que no tiene el índice como para usar.
- Junta Hash: Surge de leer 3 veces ambas tablas 3 (B(R) + B(S))

Si es la misma tabla y se utiliza el mismo atributo de junta, solo carga la tabla una vez. No tendría sentido cargar devuelta la tabla.

Pipelining

- Se concatena la salida de un operador con la entrada de otro. El resultado de un operador será la entrada de otro operador.
- Puedo llegar a hacer un ahorro en el costo (puede aumentar el uso de memoria). Por ejemplo no tener que hacer una proyección extra.
- En vez de procesar completamente un operador y luego el siguiente, puede irse procesando parcialmente el resultado a medida que se arma

Estimación de cardinalidad

- Para poder usar las fórmulas de operadores que trabajan con resultados de otros operadores, precisamos conocer la información que nos dan los metadatos
- Precisamos estimar la cantidad de filas n(R)
- Algunas operaciones modifican F(R), se recalcula B(R)

Histogramas

- Guardan la frecuencia de ciertos valores.
- Mantener actualizado el histograma es costoso.
- En general tiene una buena performance

Se trata de no utilizar operaciones en las dos tablas a la vez, se trabaja en una, se hace el join y se sigue trabajando.