Veritabanı Yönetim Sistemleri

Dr. Öğr. Üyesi Ahmet Arif AYDIN

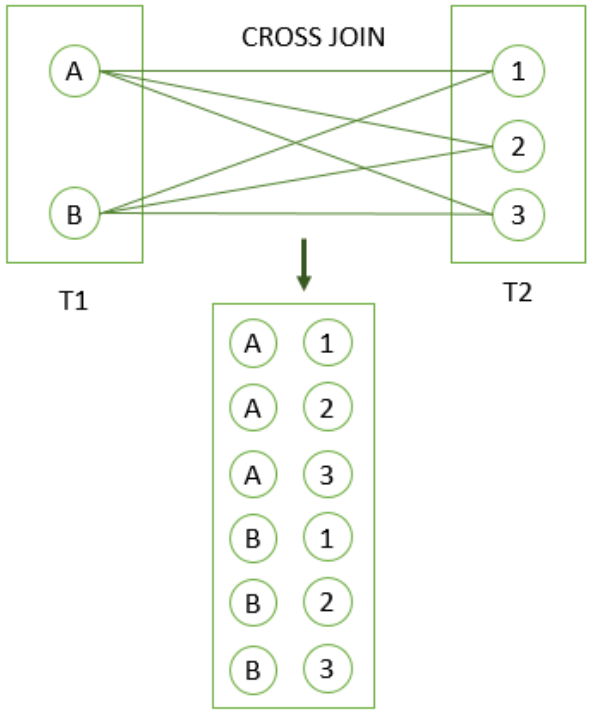
Dosya Yapıları ve İndexleme (storage & indexing)

Konular

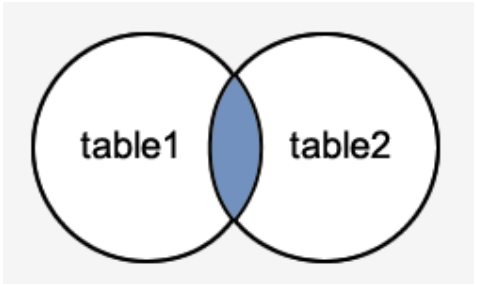
Konular
Veritabanı Tasarımı
ER-Model
UML
Relational Model
İlişkisel Cebir
İlişkisel Hesap
SQL
Dosya Yapıları ve İndexleme
Sorgu Optimizasyonu
Hareket Yönetimi
Kilitlenmeler
Concurrency
Veritabanı Kurtarma Teknikleri

Özet

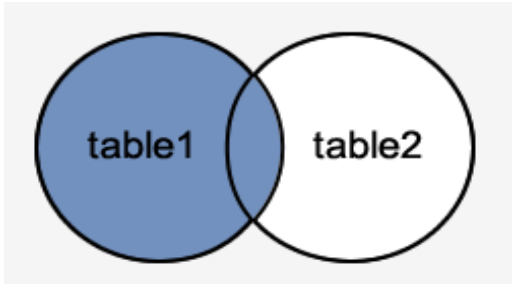
- Kısıtlamalar (Check, Unique, Not Null)



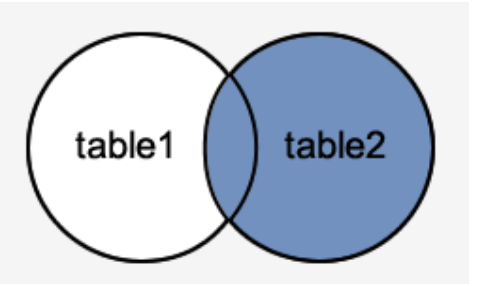
inner



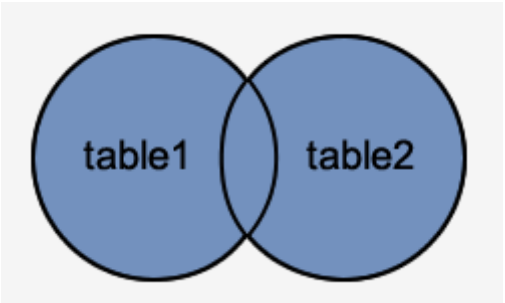
left



right



Full outer




```
SELECT * FROM T1
NATURAL [INNER, LEFT, RIGHT]
JOIN T2;
```

Veritabanı yönetimi sistemleri

- veri bütünlüğü
- veri bağımsızlığı
- verilere etkili bir biçimde erişimi
- verinin yönetimini
- veri güvenliğini

sağlamaktadır.

Dosya Yapıları ve İndexleme

Bir dosya içerisinde çalışanların **yaş , isim , adres ve maaş** bilgileri bulunsun.

- Çalışanların bilgilerine erişim için **yaş alanına göre** artan sıralama yapılabilir.
  - Eğer dosya içeriği **sıklıkla değişiyorsa ve yeni veriler ekleniyorsa** istenilen sıralamayı sağlamak maliyetli olur.
  - yaş alanından başka **diğer alanlardaki verilere erişilmek istendiğinde** istenilen verilere erişmek için **bütün kayıtların taranması** gerekmektedir.

Kayıt koleksiyonlarında bulunan birden fazla alanlara erişim

***indexing (indexleme)** tekniği ile gerçekleştirilir*

Verinin Diske Kaydedilmesi

- Veritabanı yönetim sistemleri çok çeşitli veriyi disk üzerine kaydetmektedir.
- Üzerinde işlem yapılması gereken veri hafızaya alınmaktadır.
- Verinin **diske yazıldığı** ve sonrasında ihtiyaç halinde okunduğu **her bir birim** sayfa (page) olarak adlandırılır.
- Veritabanının parametresi **sayfa (page)** dır.

PostgreSQL 'in sayfa kapasitesi **8192 bytes (8 KB)** dır.

Verinin Diske Kaydedilmesi

Her bir satırında text ve integer alanları bulunan ve 100.000 satırdan oluşan bir veri koleksiyonunu PostgreSQL de kaydedelim.

24 bytes: herbir satır başlığı (ortalama)
24 bytes: bir integer ve text
4 bytes: pointer on page to tuple

---

52 bytes (1satır)

Bir satırın kapladığı alan

8192 bytes
/
52 bytes

---

158 satır

Bir sayfada depolanacak olan Satır sayısı

100.000
/
158

633

Page (sayfa) sayısı

Verinin Diske Kaydedilmesi

Page I/O (Sayfa Giriş / Çıkış)

- Veritabaninda bulunan kayıtlara erişimi sağlamaktadır (diskten hafızaya - hafızadan diske)
- Gerçekleştirilecek sorgulama işlemlerinin maliyetini birinci dereceden etkilemektedir.
- Page I /O nun işlem maliyetlerini düşürmek için optimizasyon çalışmaları yapılmaktadır.

Verinin Diske Kaydedilmesi

- Veritabanı dosya yapısıyla beraber kaydedilen verinin istenilen sayfasını sabit maliyetle okumaya imkan sağlar
- Belirli bir sıra ile kaydedilen sayfalara erişmek, karışık olan sayfalara erişip okumaktan daha az maliyetlidir.
- Disk' de kayıtlı sayfaların her birini diğerinden ayırt etmeyi sağlayan **record id (rid)** 'si bulunmaktadır.
- **record id (rid)** sayfanın kayıtlı olduğu **adres bilgisini** içermektedir.

Ara Bellek Yöneticisi (Buffer Manager)

- Diskte kayıtlı olan veriyi hafıza getiren ve diske kayıt işlemi gerçekleştiren yazılım katmanı **buffer manager (ara bellek yöneticisi)** olarak isimlendirilir.
- **File layer(dosya katmanı)** sayfalara erişim isteğini **buffer manager'dan** ister.
- Hard disk üzerindeki alan **disk space manager (disk yöneticisi)** tarafından yönetilmektedir.
- Yeni bir **page** (sayfa)' in kaydedilmesi için File layer **disk space manager** dan alan ister.
- File layer bir page silindiğinde **disk space manager** bırakılan alanı kullanıma açık alana ekler.

Dosya Organizasyonları ve İndexleme

- Veritabanında bulunan kayıtlar **sayfalar** (page) halinde dosya (file) içerisinde saklanır.
- Bir dosya sayfalardan oluşur ve insert, delete , scan işlemlerini destekler
- File layer sayfa yönetimini sağlar.

Dosya Organizasyonları ve İndexleme

- Disk üzerinde kayıtlı olan dosyaları üzerindeki gerçekleştirilecek okuma işlemlerini optimize eder.
- Arama kriterlerine uyan sonuçlar index yardımıyla alınır.
- İndex aşağıdaki biçimde tanımlanabilir

1. key* (arama anahtarı ve veri)
2. (key , rid)
3. (key , rid-list)

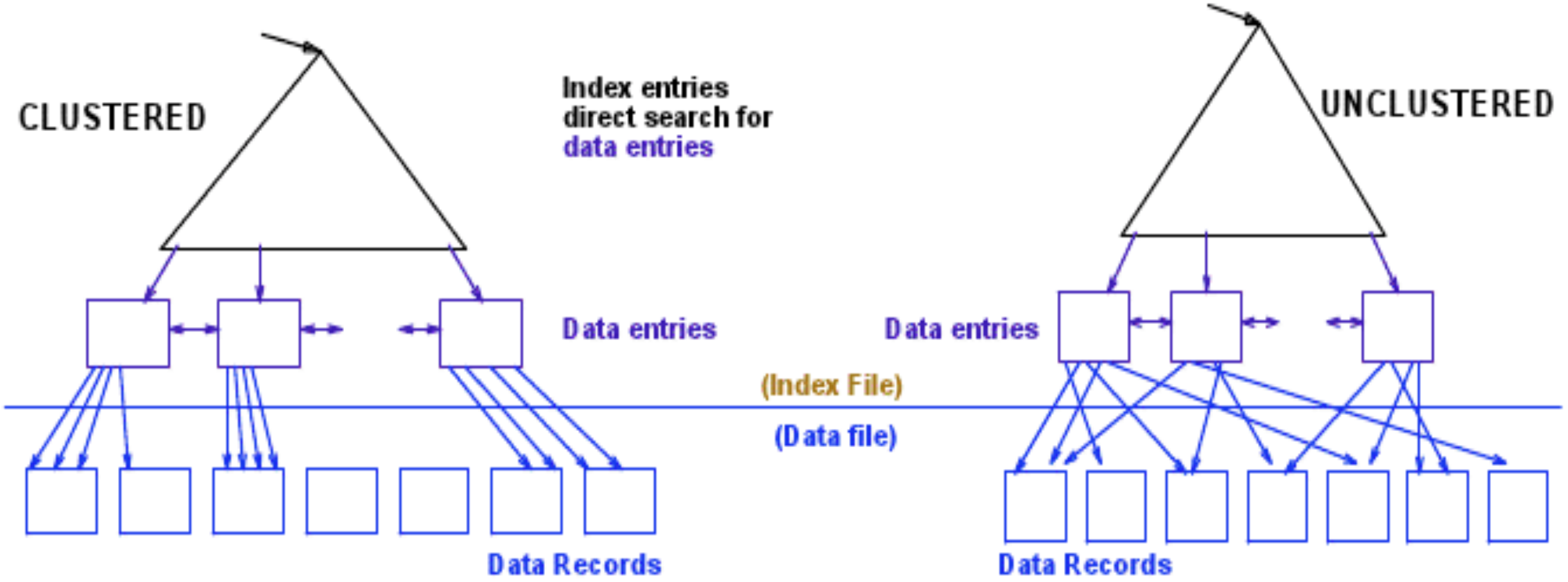
Bir dosyada bulunan kayıtların sıralanması index yapısındaki sıralamasına yakınsa bu durum **clustered index** olarak ifade edilir.

1. **key* (arama anahtari ve veri)**
2. (key , rid)
3. (key , rid-list)

- Index yapısı **clustered(kümelenmiş) index** olduğu zaman istenilen cevap birbiri ardınca sıralanan sayfa(lar) icerisindedir.

- **Unclustered** olunca bütün kayıtların incelenmesi gerekmektedir.

Dosya Organizasyonları ve İndeksleme



https://www.quora.com/How-are-clustered-and-non-clustered-indexing-implemented-internally

- Primary index
  - Tablo oluştururken tanımladığımız Primary key olan alandır
  - Unique değerler içerir
- Secondary index
  - Foreign Key
  - Unique değerler içermez

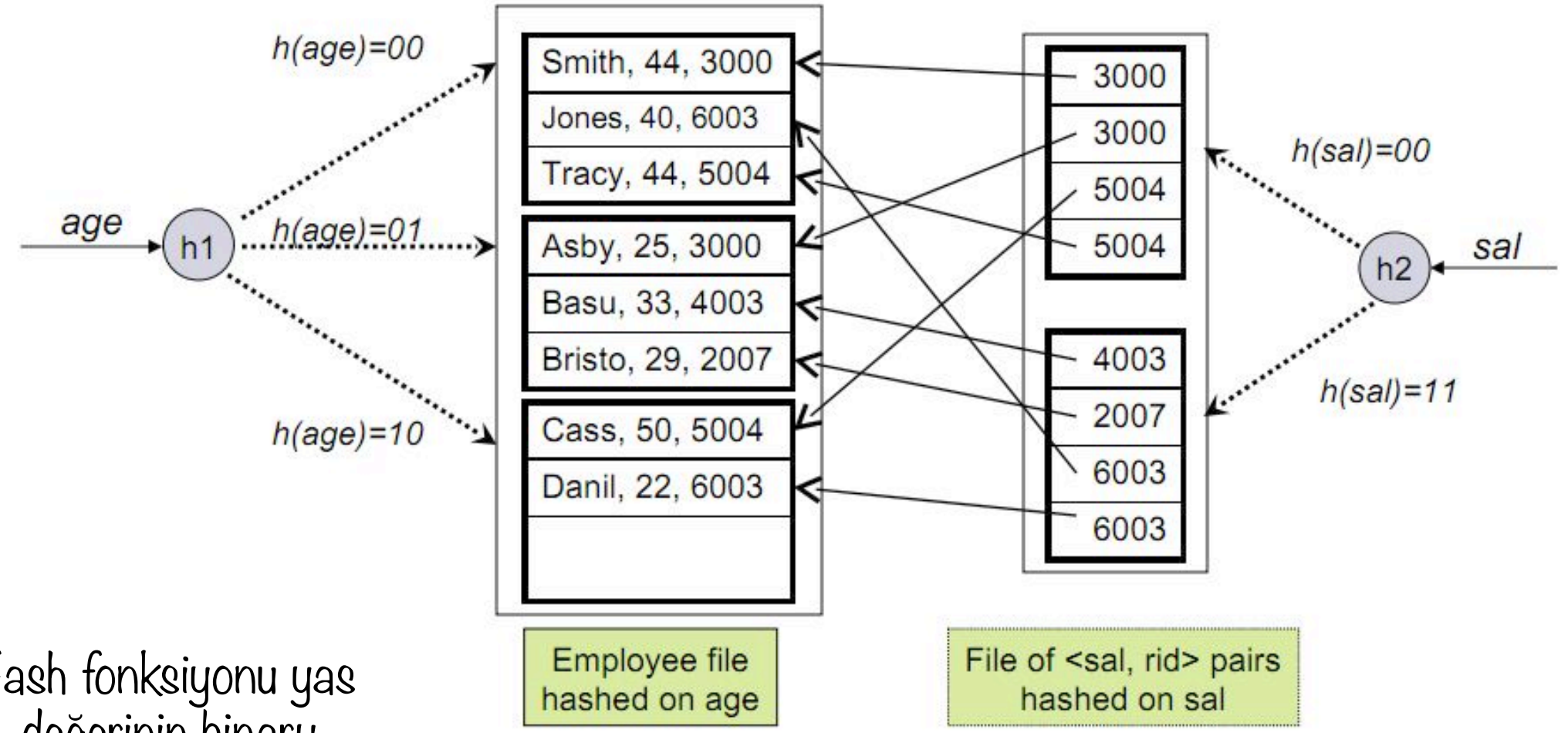
İndeks Yapısı

1. Hash data entries
2. Tree structure

İndeks Yapısı (Hash Tabanlı İndexleme)

- Kayıtlar anahtar değere göre gruplandırılır
- Her bir alt gurup **bucket** (kova) olarak isimlendirilir.
- Hash fonksiyonunun değerine göre gruplara ayırma işlemi gerçekleşir.
- Yeni değerler eklenirken hash fonksiyonuna göre bucket belirlenir.

İndeks Yapısı (Hash Tabanlı İndexleme)



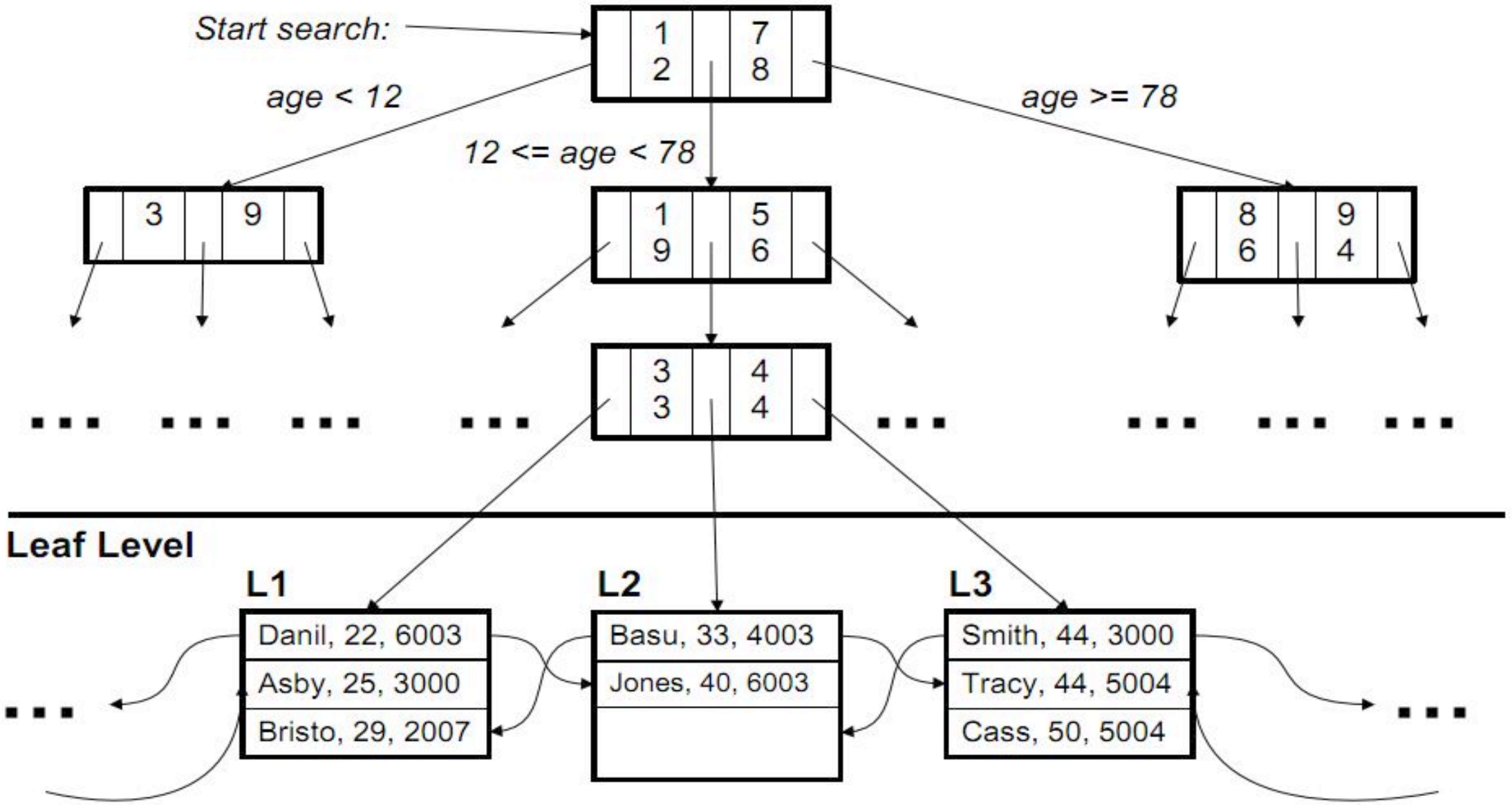
Hash fonksiyonu yas değerinin binary son iki bitine göre işlem gerçekleştiriyor

http://bosbluebluesky.blogspot.com.tr/2012/01/database-concept.html

İndeks Yapısı (Tree Based İndexleme)

- Tree based (ağaç tabanlı) indexleme hash yapısının alternatifidir.
- Belirlenen alana göre sıralama ve gruplandırma ile ağaç yapısı oluşturulur.
- Bu yapı ile beraber aramalar belirlenen aralıkta gerçekleştirilir.

İndeks Yapısı (Tree Based İndexleme)



PostgreSQL İndeks Yapıları

- B+ Tree (Balanced) — Default olarak oluşturulan index yapısıdır
- Hash — CREATE INDEX name ON table USING hash (column);
- GIST — CREATE INDEX name ON table USING gist(column);
- GIN — CREATE INDEX name ON table USING gin(column);

http://patshaughnessy.net/2014/11/11/discovering-the-computer-science-behind-postgres-indexes

İndeks Yapılarının Karşılaştırılması

Sıralı olmayan dosya (file unsorted)

- Etkili kayıt
- Hızlı tarama
- Hızlı kayıt ekleme
- Arama ve silme işlemlerinde yavaş

İndeks Yapılarının Karşılaştırılması

Sıralı dosya (sorted file)

- kayıt ekleme ve kayıt silme yavaş
- Arama sıralı olmayan file dan hızlı

İndeks Yapılarının Karşılaştırılması

Clustered file

- kayıt ekleme ve kayıt silmede etkili
- Aramalar sıralı olan dosyalardan hızlı

İndeks Yapılarının Karşılaştırılması

Hash ve Tree yapılarında

- kayıt ekleme ve kayıt silme etkili
- Aramalar hızlı
- Tarama ve aralık bulmada yavaş

İndeks Yapılarının Karşılaştırılması

File Type	Scan	Equality Search	Range Search	Insert	Delete
Heap	*BD*	0.5*BD*	*BD*	2*D*	*Search*+ D
Sorted	*BD*	*Dlog2B*	*Dlog2B +# matching pages*	*Search* + *BD*	*Search*+ *BD*
Clustered	*1.5BD*	*DlogF1.5B*	*Dlo9F1.5B+# matching pages*	*Search* + D	*Search*+ D
Unclustered tree index	*BD(R* + 0.15)	D(1 + *logFO.15B)*	*D(lo9FO.15B+# matching records)*	*D(3* + *logFO.15B)*	*Search*+ 2D
Unclustered hash index	*BD(R* + 0.125)	2D	*BD*	4D	*Search*+ 2D

B: sayfa sayısı , D: okuma yazma zamanı. C: kayıt işleme zamanı H: hash fonksıyonu,

PostgreSQL: Cascade


```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric );
```

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    shipping_address text,
    ... );
```

the child data is either deleted or updated when the parent data is deleted or updated

```
CREATE TABLE order_items (
    product_no integer REFERENCES products(product_no) ON DELETE RESTRICT,
    order_id integer REFERENCES orders(order_id ) ON DELETE CASCADE,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

PostgreSQL: Cascade


```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric );
```

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    shipping_address text,
    ... );
```

the child data is either deleted or updated when the parent data is deleted or updated

```
CREATE TABLE order_items (
    product_no integer REFERENCES products(product_no) ON DELETE RESTRICT,
    order_id integer REFERENCES orders(order_id ) ON DELETE CASCADE,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

Restrict default olarak primary key alanına bagli foreign key'ler mevcut oldugunda slime işlemini engeller.

Cascade products tablosundan bir ürün silindiğinde ilişkili tablolardan da atomatik olarak silinmesini sağlar

NO ACTION
ON UPDATE CASCADE
ON UPDATE RESTRICT

PostgreSQL: Trigger

- Bir veritabanında tanımlanan özel bir olay gerçekleştiğinde otomatik olarak veritabanında bulunan bir işlemin geröekleşmesini sağlayan prosedurdur.
- Belirli bir tabloya bir satır eklenmesi
- Bir tabloda bulunan sütunları güncellenmesi

Bir trigger'ın (tetikleyici) çalışmasını sağlar.

```
CREATE TRIGGER trigger_name
   {BEFORE | AFTER | INSTEAD OF} {event [OR ...]}
    ON table_name
    [FOR [EACH] {ROW | STATEMENT}]
    EXECUTE PROCEDURE trigger_function
```

PostgreSQL: Trigger


```
CREATE TABLE employees(
   id int4 serial primary key,
   first_name varchar(40) NOT NULL,
   last_name varchar(40) NOT NULL
);
```

```
CREATE TABLE employee_audits (
   id int4 serial primary key,
   employee_id int4 NOT NULL,
   last_name varchar(40) NOT NULL,
   changed_on timestamp(6) NOT NULL
);
```

http://www.postgresqltutorial.com/creating-first-trigger-postgresql/

https://www.youtube.com/watch?v=ewuSDIKuq4Y

PostgreSQL: Trigger


```
CREATE TABLE employees(
   id int4 serial primary key,
   first_name varchar(40) NOT NULL,
   last_name varchar(40) NOT NULL
);
```

```
CREATE TABLE employee_audits (
   id int4 serial primary key,
   employee_id int4 NOT NULL,
   last_name varchar(40) NOT NULL,
   changed_on timestamp(6) NOT NULL
);
```

```
CREATE OR REPLACE FUNCTION log_last_name_changes()
  RETURNS trigger AS
$BODY$
BEGIN
IF NEW.last_name <> OLD.last_name THEN
INSERT INTO employee_audits(employee_id,last_name,
changed_on)
VALUES(OLD.id,OLD.last_name,now());
END IF;

RETURN NEW;
END;
$BODY$
```

http://www.postgresqltutorial.com/creating-first-trigger-postgresql/

https://www.youtube.com/watch?v=ewuSDIKuq4Y

PostgreSQL: Trigger


```
CREATE TABLE employees(
   id int4 serial primary key,
   first_name varchar(40) NOT NULL,
   last_name varchar(40) NOT NULL
);
```

```
CREATE TABLE employee_audits (
   id int4 serial primary key,
   employee_id int4 NOT NULL,
   last_name varchar(40) NOT NULL,
   changed_on timestamp(6) NOT NULL
);
```

```
CREATE OR REPLACE FUNCTION log_last_name_changes()
  RETURNS trigger AS
$BODY$
BEGIN
IF NEW.last_name <> OLD.last_name THEN
INSERT INTO employee_audits(employee_id,last_name,
changed_on)
VALUES(OLD.id,OLD.last_name,now());
END IF;

RETURN NEW;
END;
$BODY$
```

```
CREATE TRIGGER last_name_changes
  BEFORE UPDATE
  ON employees
  FOR EACH ROW
  EXECUTE PROCEDURE
  log_last_name_changes();
```

http://www.postgresqltutorial.com/creating-first-trigger-postgresql/

https://www.youtube.com/watch?v=ewuSDIKuq4Y