```
create table Building (
  id int primary key,
  address varchar(5000) not null
) engine=InnoDB;

create table Apartment (
  id int primary key,
  number varchar(31) not null,
  building int not null references Building(id)
    on update cascade on delete cascade
) engine=InnoDB;

create table RoomType (
  id int primary key,
  description varchar(5000)
) engine=InnoDB;

create table RoomTypeName (
  type int references RoomType(id)
    on update cascade on delete cascade,
  name varchar(255),
  primary key(type, name)
) engine=InnoDB;

create table Room (
  id int primary key,
  area double not null,
  type int not null references RoomType(id)
    on update cascade,
  apartment int not null references Apartment(id)
    on update cascade on delete cascade
) engine=InnoDB;

create table ApartmentFloor (
  apartment int references Apartment(id)
    on update cascade on delete cascade,
  floor int,
  primary key(apartment, floor)
) engine=InnoDB;

create table Person (
  id int primary key,
```

```
  name varchar(5000) not null
) engine=InnoDB;

create table Owner (
  person int references Person(id)
    on update cascade on delete cascade,
  apartment int references Apartment(id)
    on update cascade on delete cascade,
  primary key(person, apartment)
) engine=InnoDB;
```

Part A

- A record requires an average of 6 bytes administrative overhead, including the space required for specifying the rid.
- An int uses 4 bytes.
- A double uses 8 bytes.
- A Date uses 8 bytes.
- A building address uses an average of 50 bytes.
- An apartment number uses an average of 10 bytes.
- A person name uses an average of 25 bytes.
- A room description uses an average of 100 bytes.
- A room type name uses an average of 10 bytes.
- There are 1K buildings.
- There are 20K apartments.
- There are 100K rooms.
- There are 20 room types and an average of 5 names per room type.
- The disk block size is 8K = 8192 bytes.
- A random disk access requires 6ms.
- A page can be transferred (read or write) in 0.05 ms = 50 microseconds.

Record sizes and number of blocks:

- Apartment size is 24 bytes. So 60 full blocks or 90 2/3 blocks.
- Building size is 60 bytes. So about 8 full blocks or about 12 2/3 blocks.
- RoomTypeName size is 20 bytes. So it fits in one block.
- RoomType size is 110 bytes. So it fits in one block. This is not used.
- Room size is 26 bytes. So 325 full blocks or 488 2/3 blocks.

1. Find all living rooms (room type is named "Living Room" for which one may assume there is only one room type).

   This is two separate queries. First find the room type identifier, then find the living rooms with this type. All structures require one random access to find the room type identifier because it fits in one block. This is 6ms.

   (a) Clustered Heap. One random access for the first block, then 324 sequential accesses. 6ms + 6ms + 16ms = 28ms or about 30ms.

   (b) Clustered Sort. Assuming sorted by type, it takes an average of 162 sequential accesses to get to the room type. 6ms + 6ms + 8ms = 20ms.

   (c) Clustered B-tree. There are 20 room types and 100K rooms. So there will be 5K rooms which requires about 24 blocks. Two random accesses are required for the first block, then 23 random accesses for the rest. So 26 * 6ms = 156ms or about 160ms.

   (d) Unclustered heap with index. A single random access gives the first block of pointers. Additional random accesses will be needed for the other blocks of pointers. However, these are not significant. Each disk pointer requires a random access, and there are 5K such pointers. This yields 5K * 6ms = 30K ms or about 30 seconds.

2. Find all apartments at "100 Main Street".

   This is two separate queries. First find the identifier of the building at the specified address, then find the apartments at that building. There will be 20 apartments on the average. So they fit in a single block. If there are multiple buildings, then simply repeat the second query for each building identifier.

   First the building query part:

   (a) Clustered Heap. One random access for the first block, then 7 sequential accesses. 6ms + 0.35ms or about 6ms.

   (b) Clustered Sort. Same as heap but half as many sequential accesses. Still about 6ms. One could also use a binary search. Since 8 is 2¡sup¿3¡/sup¿, this would require 18ms.

(c) Clustered B-tree. Two random accesses or about 10ms.

(d) Unclustered heap with index. Two random accesses or about 10ms.

Next the apartment query part:

(a) Clustered Heap. One random access for the first block, then 59 sequential accesses. 6ms + 3ms = 9ms or about 10ms.

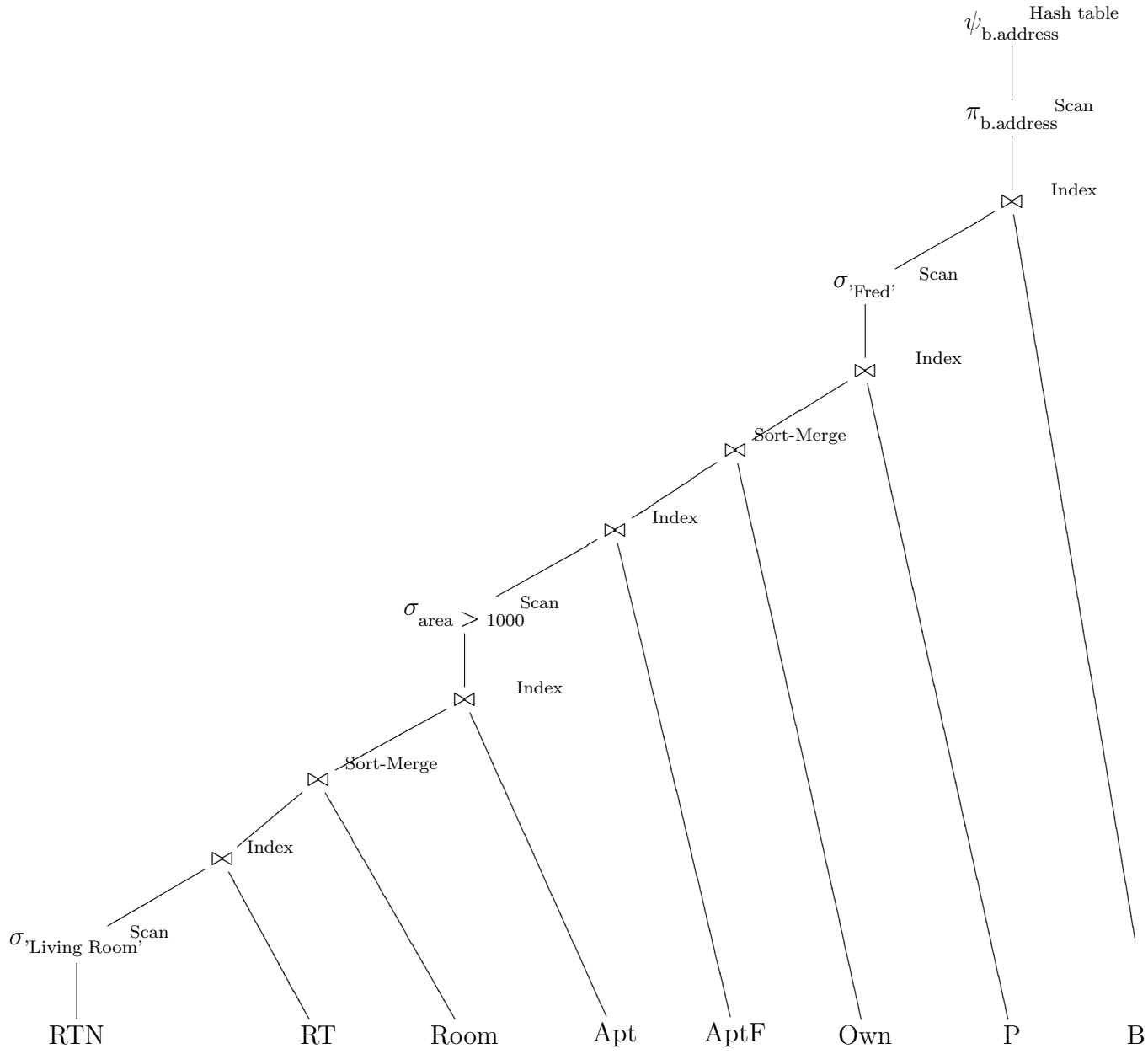
(b) Clustered Sort. Same as heap but half as many sequential accesses. Still about 10ms. One could also use a binary search. Since 60 is about 2¡sup¿6¡/sup¿, this would require 36ms.

(c) Clustered B-tree. Two random accesses or about 10ms.

(d) Unclustered heap with index. Two random accesses or about 10ms for each apartment or about 200ms in all.

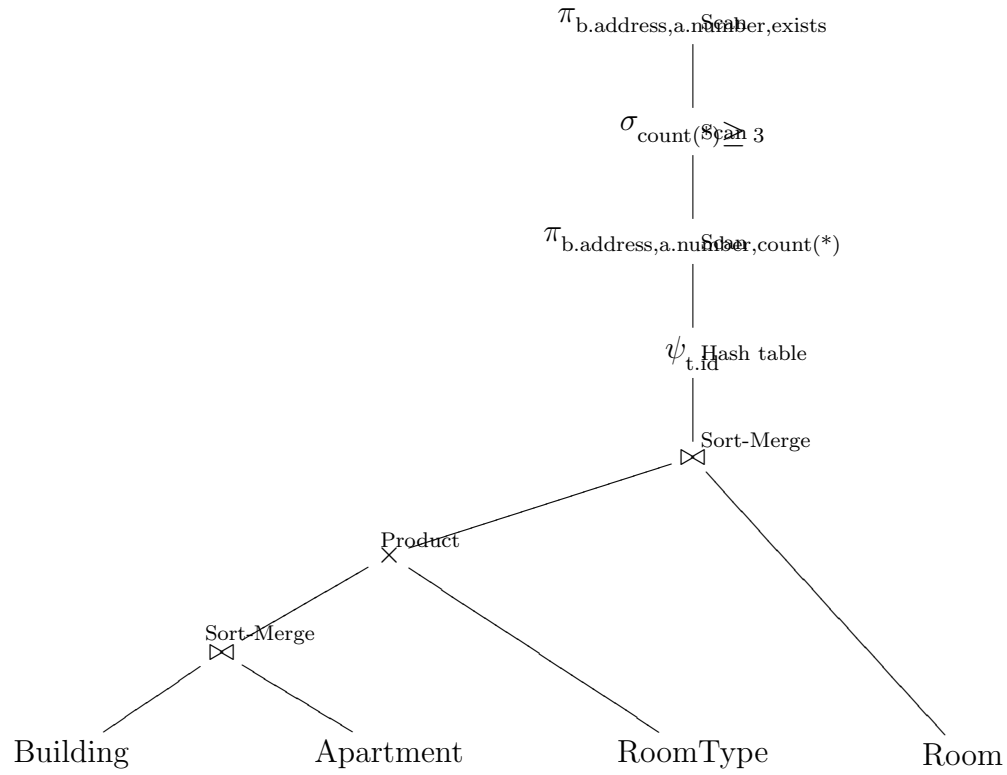
Part B

```
select distinct b.address
  from Apartment a, Building b, ApartmentFloor f, RoomTypeName n, Owner o, Person p, Roo
 where a.building = b.id
   and f.apartment = a.id
   and f.floor >= 5
   and n.type = t.id
   and n.name = 'Living Room'
   and o.apartment = a.id
   and o.person = p.id
   and p.name = 'Fred'
   and r.type = t.id
   and r.area > 1000
   and r.apartment = a.id
```




```
select b.address, a.number
  from Building b, Apartment a
 where b.id = a.building
   and exists (
         select *
           from RoomType t
          where 3 <= (select count(*)
                        from Room r
```

```
                    where r.type = t.id
                      and r.apartment = a.id)
      )
```




```
select b.address, count(*)
  from Building b, Apartment a
 where b.id = a.building
 group by b.id
 order by b.address
```