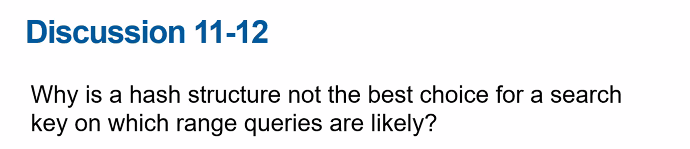
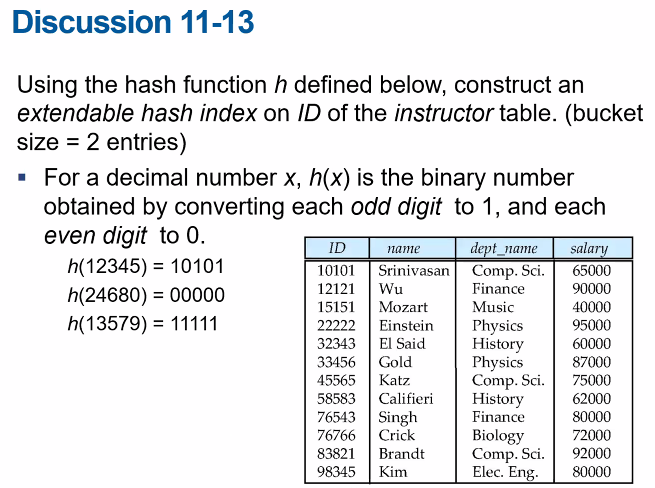
**Discussion 05/25**



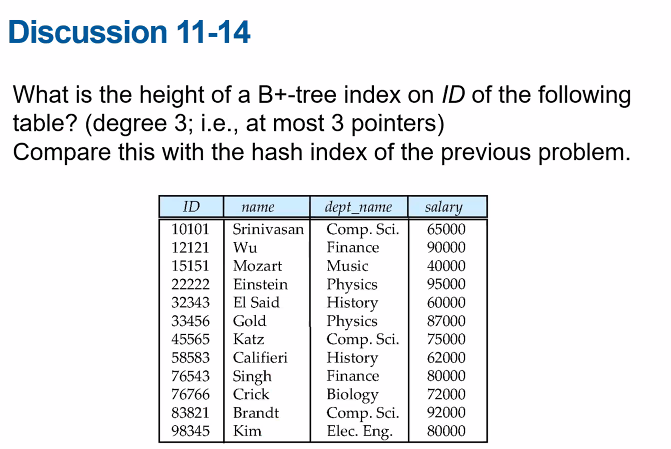
Hash structure에서는 인접한 값이라도 저장되는 위치가 인접해있다는 보장이 없기 때문에 Range query를 수행할 때는 결국 해당 range에 해당하는 모든 값을 찾으려면 저장되어 있는 모든 record를 다 돌아봐야 한다. 따라서 적합하지 않다.

=> Hash function의 특징이 locality를 없애는 것이기 때문에 (uniform, random). Locality를 지원하면서 hash를 짜면 hash의 장점이 없어진다.



처음 3개 insert => Srinivasan과 Wu가 모두 hash function이 10101임. Mozart는 11111. Hash prefix는 2이고 address table entry의 개수는 4개. Bucket 1에는 Srinivasan과 Wu이고 i\_1 = 2. Bucket 2에는 Mozart가 있고 i\_2 = 2.

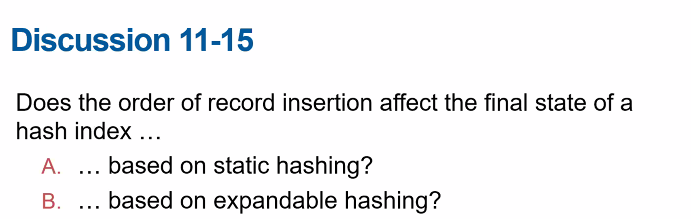
**전체 insert는 각자 해보기!**



10101 12121 – 15151 22222 – 32343 33456 – 45565 58583 – 76543 76766 – 83821 98345가 leaf node이고 그 위에는 12121 22222 – 33456 58583 – 76766 98345. 그 위에는 22222 58583 – 98345. Root는 58583.

Height는 4.

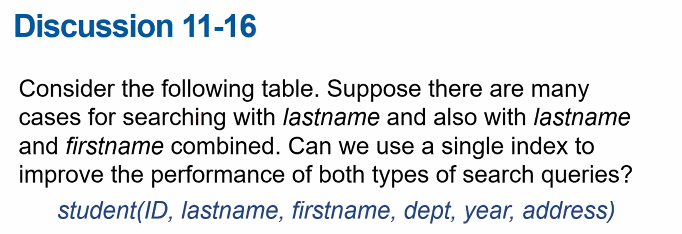
Search할 때, hash는 평균적으로 address 버켓+ 해당 버켓 해서 2개인데 얘는 평균적으로 4개의 block을 access 해야 함. 개수가 많아질수록 차이가 커짐.



A. static hashing은 hash function이 정해진 bucket address의 집합에 일정하게 매핑하므로, insert의 순서는 상관이 없을 것 같다.

B. expandable hashing은 순서가 상관이 없을 것 같은데 이유를 모르겠음…

=> split되는 정책이 일관성이 있다면, mapping되는 address는 순수하게 hash function에 의해서 결정이 되는 것이고, split 되느냐 마느냐는 테이블의 record의 분포에 따라 되는 것이기 때문에 **순서가 변해도 바뀌는 것은 없다.**



lastname으로 index를 만들면 두 가지 모두 성능이 향상될 수 있다. 우선 lastname 단독은 자명함. 그리고 lastname + firstname의 경우 lastname으로 해당하는 record들의 범위를 줄이고, 줄어든 범위에서만 firstname을 기반으로 찾으면 되므로 성능이 향상된다.

=> B+ 트리 인덱스는 prefix에 대한 index를 support할 수 있음. [lastname + firstname] 으로 index를 만들면 prefix에 대한 search가 되기 때문에 lastname, lastname + firstname search의 효율이 모두 향상된다.

Hash의 경우는 다름. Hash function은 [lastname + firstname] 에서 lastname이 같다고 같은 맵핑이 되지가 않음.

실제로 DBA들이 index로 search performance를 올리기 위해서 composite index를 많이 사용함. 주로 search가 되는 attribute를 앞에 두고, specific하게 search할 때 이용되는 것들을 뒤에 놓음.