Midterm Review

CS 143 Introduction to Database Systems

TA: Jin Wang

02/07/2020

Outline

- General Information
- Basic Concepts
- Relational Algebra
- SQL
- Data Storage and Indexes

General Information

- Time: Feb 11st, 4:00 – 5: 50 PM
- Location: Haines Hall A2
- Scope:
  - Everything in lecture from week 1 to 5, cut-off to index
  - **But also include Nested Loop Join, Index Join**
- Type of questions:
  - Free response
  - Contents and formats are very close to those in lecture yesterday.
- Policy
  - Close Book, Close Notes
  - Allow 1 page Cheat sheet

Basic Concepts

- Data Model
- Relational Data Model
  - Query Language: DML, DDL
  - Schema, Table, Tuple, Attribute
  - Super key, primary key, foreign key
- Referential Integrity

Referential Integrity: example

```
CREATE TABLE Department (dept_id INT NOT NULL,
                         dept_name VARCHAR(256),
                         PRIMARY KEY (dept_id)) ENGINE=INNODB;


CREATE TABLE Employee (emp_id INT NOT NULL,
                       emp_name VARCHAR(256),
                       dept_id INT,
                       FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
                       ON DELETE CASCADE) ENGINE=INNODB;
```

Relational Algebra

- Way to ask queries of relations
- Operators

Operation	Symbol	Explanation
Selection	σ	Selects rows
Projection	π	Selects columns
Union	∪	
Difference	-	
Cross-product	×	
Join	⋈	Join tables on some condition
Rename	ρ	
Aggregate	ϑ	

Relational Algebra Practice

Consider the following schema about students and courses. Primary keys are underlined.

Students (<u>sid</u>, sname, street, city, age, gender)
Registered (<u>sid</u>, <u>cid</u>, grade)
Courses (<u>cid</u>, cname, profname)

a) In the space below, write a Relational Algebra expression that returns the sid and sname of all students who received an “A” in a course taught by “Hilfinger”.

$$\pi_{\text{sid,sname}}(\sigma_{\text{grade='A'} \wedge \text{profname='Hilfinger'}}(S \bowtie R \bowtie C))$$

Schema
Students (sid, sname, street, city, age, gender)
Registered (sid, cid, grade)
Courses (cid, cname, profname)

b) In the space below, write a Relational Algebra expression that returns the sid of all students who have taken both CS162 and CS186 (where CS162 and CS186 are “cid”s) but no other courses. Do not use any unnecessary relations.

$$[\pi_{sid}(\sigma_{cid='CS162'}(R)) \cap \pi_{sid}(\sigma_{cid='CS186'}(R))] - \pi_{sid}(\sigma_{cid\neq'CS162' \wedge\ cid\neq'CS186'}(R))$$

Schema
Students (sid, sname, street, city, age, gender)
Registered (sid, cid, grade)
Courses (cid, cname, profname)

Objective Questions

[5 points] Consider two relations with the same schema: R(A,B) and S(A,B). Which one of the following relational algebra expressions is not equivalent to the others?

$\Pi_{R.A}((R \cup S) - S)$

$\Pi_{R.A}(R) - \Pi_{R.A}(R \cap S)$

$\Pi_{R.A}(R - S) \cap \Pi_{R.A}(R)$

2nd is different!
(Set diff on R.A)

Can intersection be expressed using set difference?

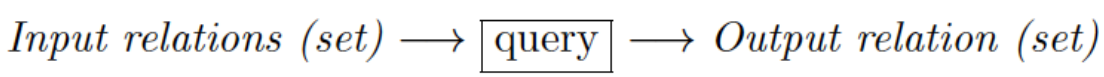
Yes: $R \cap S = R - (R - S) = S - (S - R)$

Can intersection be expressed using cartesian product and projection?

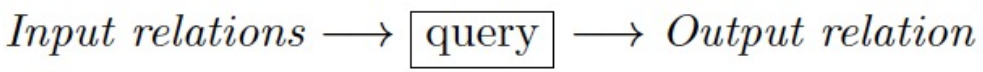
No, neither operator can remove elements!

Relational Algebra vs. SQL

- Relational Algebra
  - set semantics, no duplicate tuples



- SQL
  - multiset semantics
  - Need to explicitly add keyword DISTINCT for deduplication



RA vs. SQL: sample question

1. Consider the following SQL query on the table R(A, B, C). You may assume that there are no NULLs.

```
SELECT R1.A
FROM   R R1, R R2
WHERE  R1.A=R2.A AND R1.B='UCLA' AND R2.C='blue';
```

Is it possible to write an equivalent relational algebra expression? If yes, write such an expression succinctly. If no, briefly explain why.

ANSWER:

No. When we project on A, there can be multiple tuples that satisfy the condition with the same A values. A relational algebra expression will always remove such duplicates while the above SQL query will not.

SQL Basics

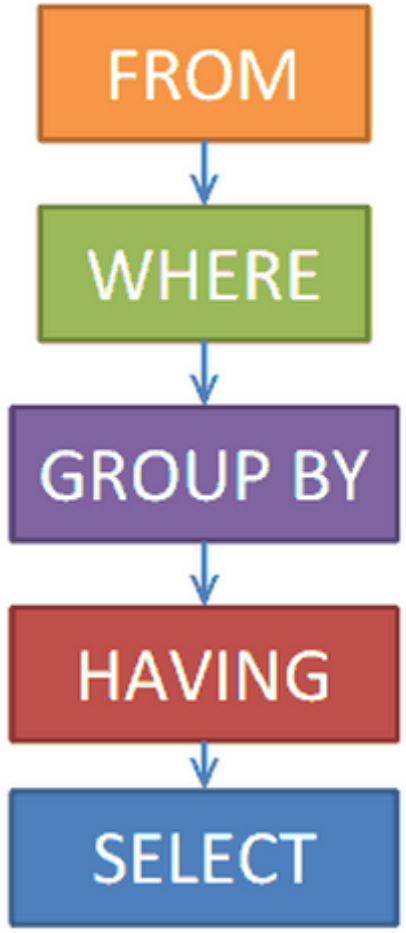
SELECT **target-list**
FROM **relation-list**
WHERE **qualification**

target-list: list of attributes in each relation (or *)
relation-list: list of relations, usually a table
qualification: set of select clauses

SQL Basics

- SELECT standing, gpa, COUNT(*)
  FROM Students
  WHERE sname STARTS_WITH ‘A’
  GROUP BY standing, gpa
  HAVING COUNT(*) > 3;

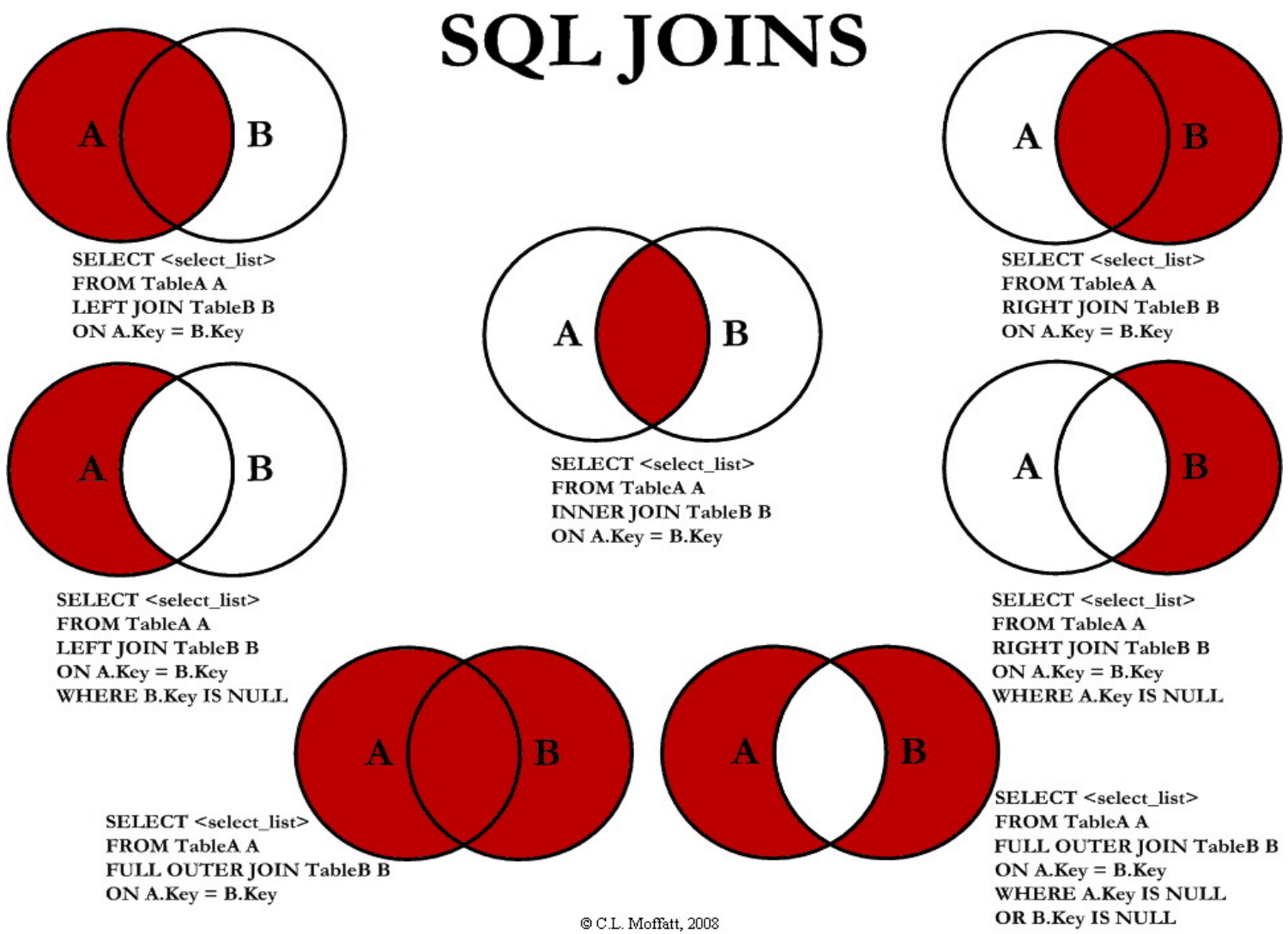
  Where does aggregation happen?
  During GROUP BY!



SQL basics

- You should be familiar with the usage of following keywords
  - relation: EXCEPT, UNION, INTERSECT
  - sub-query: (NOT) IN, EXIST
  - join: OUTER/INNER, LEFT/RIGHT, ON
  - aggregation: GROUP BY, HAVING, COUNT
  - alias of tables

Join



Recall this figure: different types of SQL join

Nested Queries: recall this example

```
1  SELECT
2      AVG(end_time - start_time) / 60 AS avg_trip_length_mins
3  FROM (
4      SELECT trip_id, user_id, start_time, end_time
5      FROM (
6          SELECT
7              l.trip_id AS trip_id,
8              l.user_id AS user_id,
9              l.time AS start_time,
10             r.time AS end_time
11         FROM start l
12         LEFT JOIN end r
13         ON l.trip_id=r.trip_id AND l.user_id=r.user_id
14     ) my_subquery
15 ) joined;
```

Think of this as a new table that is created from which you are now querying.

Note that all derived tables must have an alias or you will get an error.

SQL Practice I

Recall the schema about students and courses from the previous question:

Students (<u>sid</u>, sname, street, city, age, gender)
Registered (<u>sid</u>, <u>cid</u>, grade)
Courses (<u>cid</u>, cname, profname)

Note that all of the SQL queries in parts a and b (next two slides) are syntactically valid

a) Which of the following queries produces the CIDs of courses that have no registered students? (One or more options are correct)

A. SELECT c.cid
FROM Courses c LEFT OUTER JOIN Registered r ON c.cid = r.cid
HAVING COUNT(*) > 0;

B. SELECT cid FROM Registered
EXCEPT
SELECT cid FROM Courses;

C. SELECT cid FROM Courses
WHERE cid IN
(SELECT cid FROM Registered
GROUP BY cid HAVING COUNT(*) = 0);

D. SELECT c.cid FROM Courses c
WHERE NOT EXISTS
(SELECT cid FROM Registered r WHERE r.cid = c.cid);

E. None of the above

Answer: D

Schema
Students (sid, sname, street, city, age, gender)
Registered (sid, cid, grade)
Courses (cid, cname, profname)

b) Which of the following queries are equivalent to the query: SELECT DISTINCT profname FROM Courses (One or more options are correct)

A. SELECT profname FROM Courses GROUP BY profname;

B. SELECT profname FROM Courses
UNION SELECT profname FROM Courses;

C. SELECT DISTINCT profname FROM Courses
UNION ALL SELECT profname FROM Courses

D. SELECT DISTINCT profname FROM Courses WHERE NULL = NULL

E. None of the above

Schema
Students (sid, sname, street, city, age, gender)
Registered (sid, cid, grade)
Courses (cid, cname, profname)

Answer: A, B

c) When would the following two queries return different results for a given database instance? A one sentence answer should be sufficient!!!

```
SELECT s.sname FROM Students s LEFT OUTER JOIN Registered r
  ON s.sid = r.sid
SELECT s.sname FROM Students s, Registered r WHERE s.sid =
  r.sid
```

when a student is not registered for a course

Schema
Students (sid, sname, street, city, age, gender)
Registered (sid, cid, grade)
Courses (cid, cname, profname)

d) In the space below, write a SQL query that returns the name and SID of every student enrolled in the class 'CS186' whose age is greater than the average age of all the students enrolled in that class. ('CS186' is a CID.)

One solution with subqueries:

```
SELECT S.sname, S.sid
FROM Students S, Registered R
WHERE S.sid = R.sid AND R.cid = 'CS186' AND
        S.age > (SELECT AVG(S2.age)
                 FROM Students S2, Registered R2
                 WHERE S2.sid = R2.sid AND R2.cid = 'CS186')
```

Schema
Students (sid, sname, street, city, age, gender)
Registered (sid, cid, grade)
Courses (cid, cname, profname)

SQL Practice II

Assume the following tables for this problem:

Employee (person-name, age, street, city)
Work (person-name, company-name, salary)
Company (company-name, city)
Manage (person-name, manager-name)

A person's name is unique, but a person may work for more than one company. A company name is unique, but a company may be located in more than one city.

a) Write a query in SQL to find the names of such companies that all of their employees have salaries higher than $100,000.

```
SELECT company-name
    FROM Company C
    WHERE 100000 < ALL
                (SELECT salary FROM Work W
                 WHERE C.company-name = W.company-name)
```

Schema
Employee (person-name, age, street, city)
Work (person-name, company-name, salary)
Company (company-name, city)
Manage(person-name, manager-name)

b) Find the name(s) of the employee(s) whose *total* salary is higher than those of all employees living in Los Angeles

```
SELECT person-name FROM Work
GROUP BY person-name
HAVING SUM(salary) > ALL
        (SELECT SUM(salary) FROM Work, Employee
         WHERE Work.person-name = Employee.person-name AND city='Los Angeles'
        GROUP BY Work.person-name)
```

Schema
Employee (person-name, age, street, city)
Work (person-name, company-name, salary)
Company (company-name, city)
Manage(person-name, manager-name)

c)* Find the name(s) of the manager(s) whose *total* salary is higher than that of at least one employee that they manage.

```
Total_Salary:
SELECT person-name, SUM(salary) total-salary
FROM Work
GROUP BY person-name

SELECT  manager-name
FROM  Manage  M
WHERE  EXISTS
  (SELECT * FROM
          Total_Salary  S1,
          Total_Salary  S2
   WHERE  M.manager-name=S1.person-name AND
             M.person-name = S2.person-name AND
             S1.total-salary > S2.total-salary)
```

Schema
Employee (person-name, age, street, city)
Work (person-name, company-name, salary)
Company (company-name, city)
Manage(person-name, manager-name)

Constraint

- Overview
  - specify rules for the data in a table.
  - can be applied either column level or table level
- Common Constraints
  - **NOT NULL** - Ensures that a column cannot have a NULL value
  - **UNIQUE** - Ensures that all values in a column are different
  - **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
  - **FOREIGN KEY** - Uniquely identifies a row/record in another table
  - **CHECK** - Ensures that all values in a column satisfies a specific condition
  - **DEFAULT** - Sets a default value for a column when no value is specified
  - **INDEX** - Used to create and retrieve data from the database very quickly

Constraint: sample question

Consider a relation $T(A, B)$. For each of the three SQL assertions below, write in the space provided a *plain English* description of the constraint that is enforced by the assertion. Please be specific about the actual attributes involved in the constraint. In all three cases the correct answer corresponds to a single concept from the course and can be specified in a few words.

(a)
```
CREATE ASSERTION First CHECK(
  NOT EXISTS(SELECT *
             FROM T T1, T T2
             WHERE T1.A <> T2.A))
```

ANSWER:
Every non-NULL A in T must have the same value.

(b)
```
CREATE ASSERTION Second CHECK(
  NOT EXISTS(SELECT B
             FROM T
             WHERE B NOT IN (SELECT A FROM T)))
```

ANSWER:
B is a foreign key referencing A.

View

- Definition
- Characteristics of view
  - Do not hold data
  - When corresponding tables are updated, views are also updated
  - Definition of views not stored together with data
- Update on a view
  - FROM: only one table
  - SELECT: only column names
  - Attributes not in SELECT allow NULL values
  - NO GROUP BY or HAVING

Trigger

- Definition
  - A special stored procedure that is run when specific actions occur within a database
  - Characteristics: when, what
- Example

```
CREATE TABLE employees_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employeeNumber INT NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

```
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW
BEGIN
    INSERT INTO employees_audit
    SET action = 'update',
     employeeNumber = OLD.employeeNumber,
        lastname = OLD.lastname,
        changedat = NOW();
END$$
```

A trigger defined for the update operations

Authorization

- Application Scenario
- Process of grant
- Keywords
  - GRANT
  - REVOKE
  - CASCADE
  - RESTRICT

A template for granting privileges:

```
GRANT
    list_of_privileges
 ON
    table_or_view_name
 TO
    user_or_role_list     -- comma separated
```

A is the owner of the table `R`. Users of our database system have executed the following sequence of commands:

1. By User A: `GRANT SELECT ON R TO` B, E `WITH GRANT OPTION;`
2. By User B: `GRANT SELECT ON R TO` C `WITH GRANT OPTION;`
3. By User C: `GRANT SELECT ON R TO` D `WITH GRANT OPTION;`
4. By User E: `GRANT SELECT ON R TO` C`;`
5. By User A: `REVOKE GRANT OPTION FOR SELECT ON R FROM` B `CASCADE;`

In the following table, indicate whether each user has the `SELECT` privilege on `R` with/without `GRANT OPTION`. If the user has the corresponding privilege write YES in the corresponding entry.

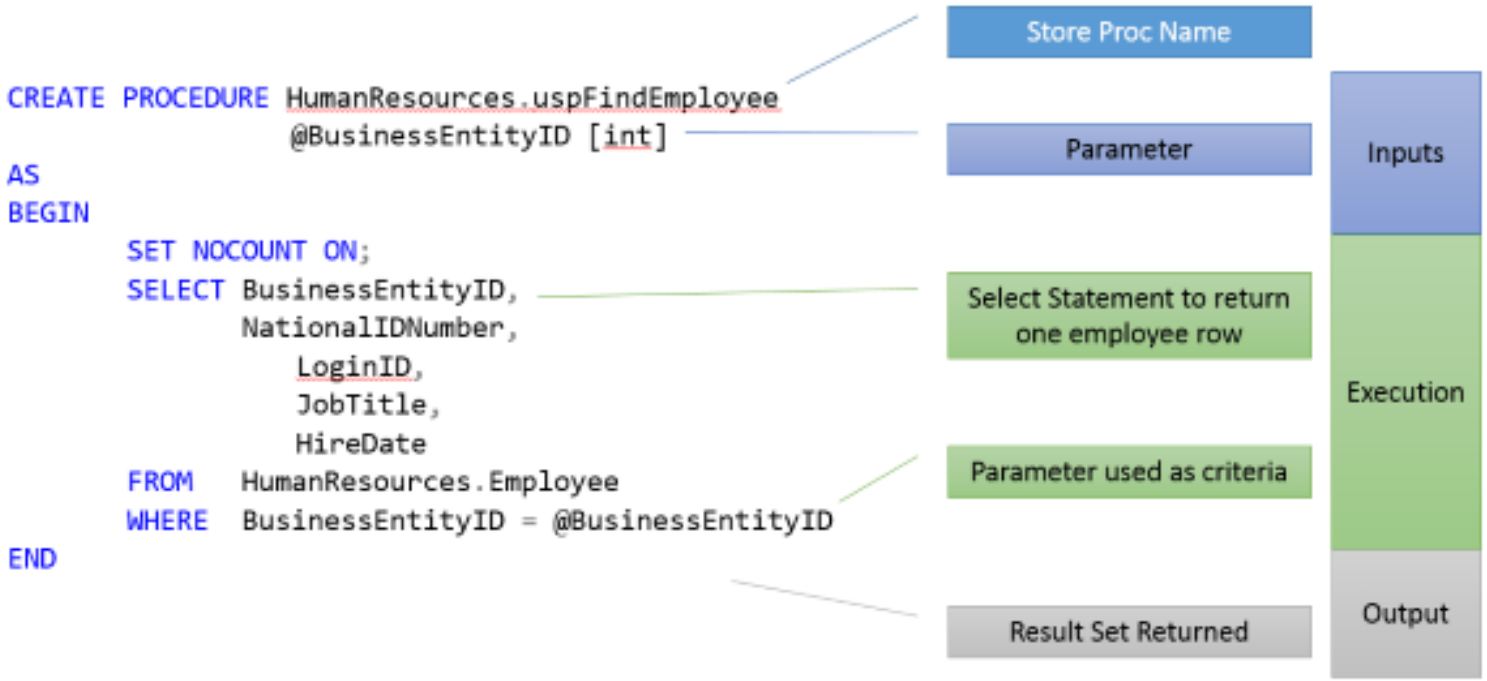
User	SELECT ON R?	GRANT OPTION?
A		
B		
C		
D		
E		

ANSWER:

A: YES, YES. B: YES, NO. C: YES, NO. D: NO, NO. E: YES, YES.

Procedure

- Definition
  - A piece of prepared SQL code can be reused again
  - Executed by the CALL struct
- Example



Data Storage

- Disk
  - I/O: sequential vs. random
  - Be able to make simple calculation
  - Organize files in a block

B+ Tree Index

- Concepts about Index
  - Dense/Sparse
  - Clustered/non-clustered
  - Primary/Secondary
- Design goal of B+ Tree: reduce the number of I/O
- Operations
  - Insert
  - Delete
  - Point Query/Range Query
  - Update

Primary vs. Secondary
Clustering vs. Non-clustering

- Primary index
  - Clustering
  - Tuples in the table are ordered by the index search key
- Secondary index
  - Non-clustering
  - Tuples in the table are not ordered by the index search key
    - Index on a non-search-key for sequential file
    - Unordered file

A sparse index can only be built on clustered data and should be primary index.

Primary Index vs. Secondary Index

Sequential File

10	
20	

30	
40	

50	
60	

70	
80	

90	
100	

30	
50	

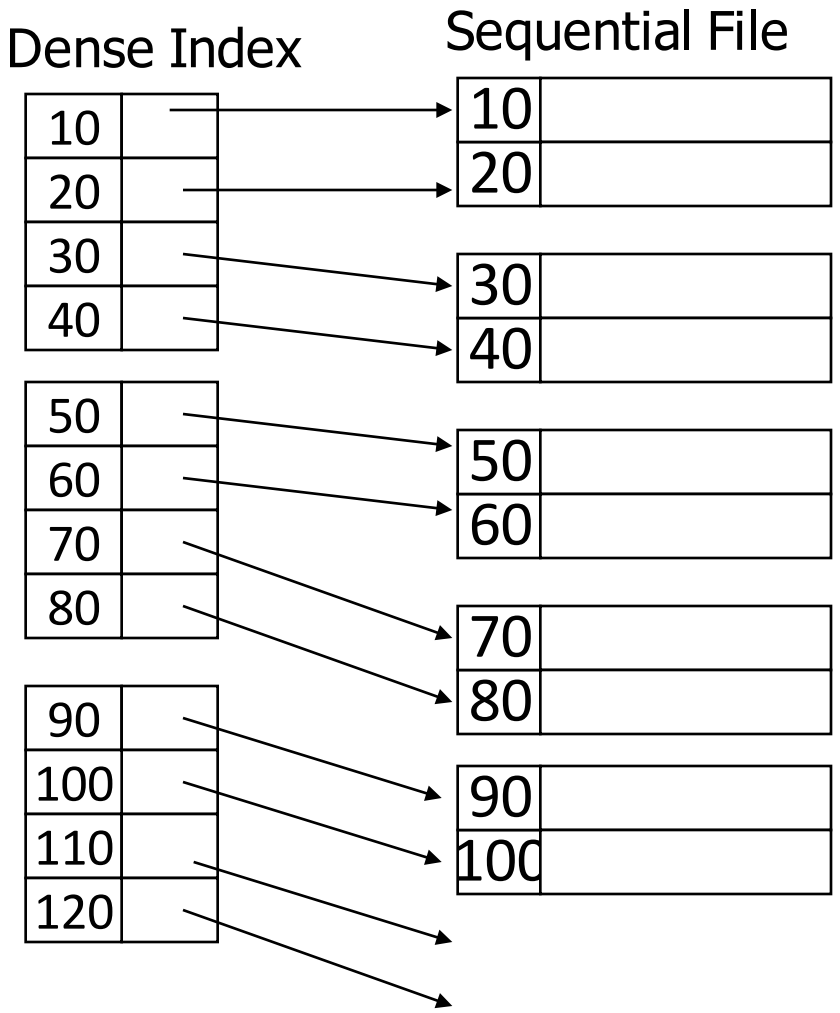
20	
70	

80	
40	

100	
10	

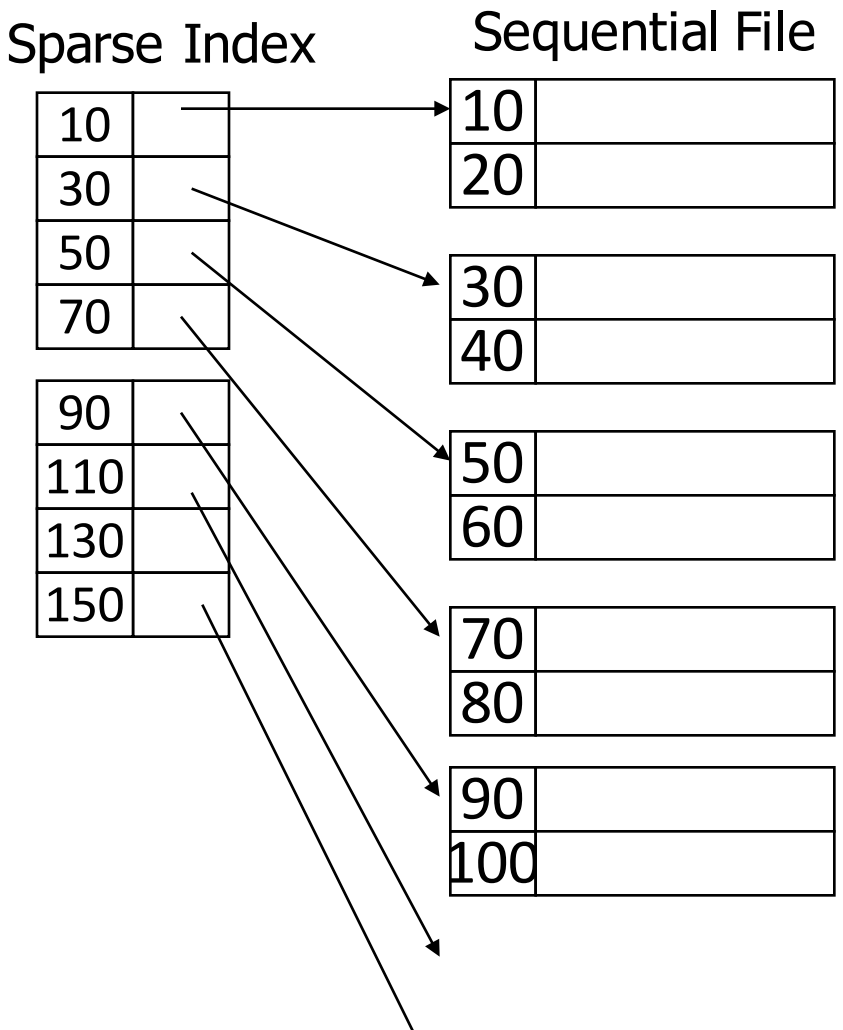
90	
60	

Dense, Primary Index



- Primary index (clustering index)
  - Index on the search key
- Dense index
  - (key, pointer) pair for **every record**
- Find the key from index and follow pointer
  - Maybe through binary search
- Q: Why dense index?
  - Isn't binary search on the file the same?
  - Index require less space and could be load into memory

Sparse, Primary Index



- Sparse index
  - (key, pointer) pair per **every "block"**
  - (key, pointer) pair points to the first record in the block
- Q: How can we find 60?
  - Binary search on sparse index - 50
  - Load the block
  - Find 60

B+ Tree Index: sample question

We are using a B+ tree with actual data records in leaf pages to store one billion records.
Each records is 200 bytes, each disk page has 16KB (16,384 Bytes) and will always be at most 67% full.

1. How many leaf pages are required?

16384 * 0.67 / 200 = ~54 Entries per page. 10^9 / 54 = ~18.5 * 10^6 pages.

2. Assume each index entry takes 32 bytes. What is the maximum fanout of the index?

16384 * 0.67 / (32) = ~343

3. What is the minimum height of the tree? Assume there is enough space in the leaf page,
how many I/O operations are required to insert a new record?

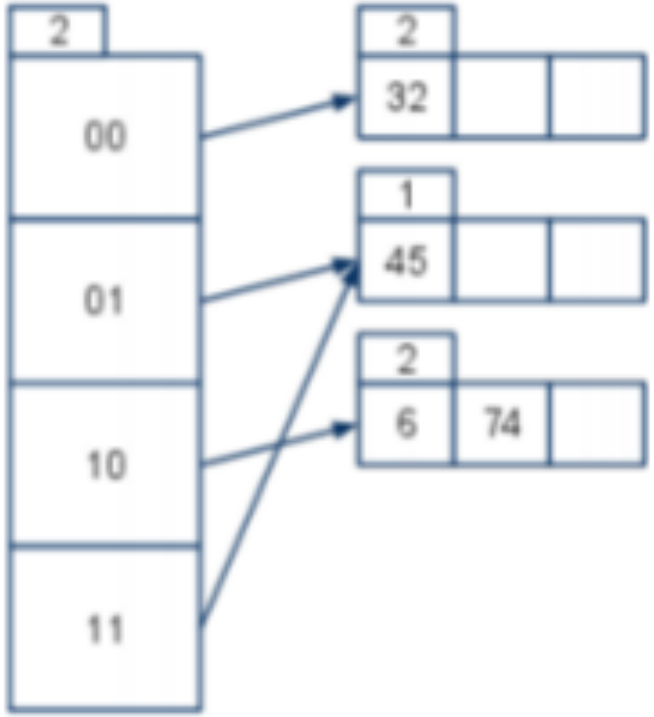
Minimum Height = log_343(18.5 * 10^6)+1 = ~3+ 1 = 4
We need 4 Reads (3 non-leaf reads, 1 leaf read) + 1 Write = 5 I/Os.

Hash Index

- Unlike B+ Tree, do not support range query
- Static hash
  - Use overflow pages
- Extendible hash
  - Use i of b bits from outputs, increase i on demand
  - Accelerate searching: use directory
  - Operations: add/delete

Hash Index: sample question

Consider the Extendible Hashing structure below. What is the maximum number of keys can you insert before the size of the directory must double?

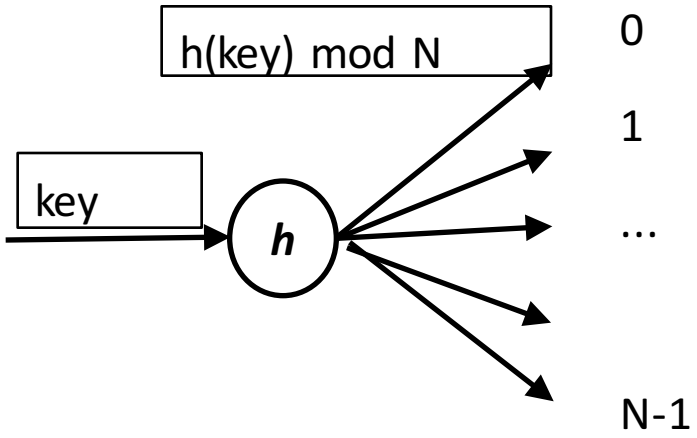


Answer: 8

Trees vs. Hashing

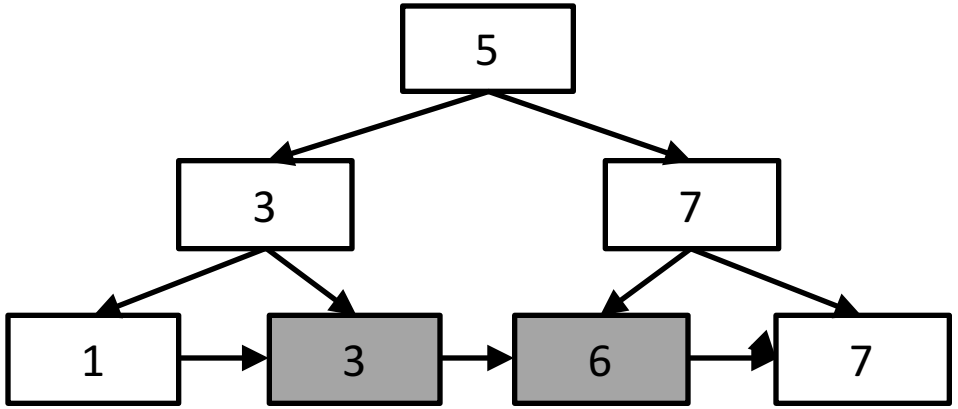
- Hashes good at:
  - *Equality*
  - *k == 1*
- Because:

Ideally one lookup!



- Trees good at:
  - *Range Queries*
  - *3 <= k <= 6*
- Because:

Data laid out to scan!



Good Luck for your midterm!

Thank you!