CS 186 Section 3: SQL, Joins, and Files

Vikram Sreekanti

Multi-Table SQL Queries

Multi-Table SQL

- **Main idea: These queries are very similar to single table queries.**
- Main differences:
  - There will be multiple tables listed in the `FROM` clause.
  - You (usually) specify a “join predicate”.
    - What is a join predicate?
      - A predicate on a value common to both tables.
        - e.g. `Students.sid = Enrollment.sid`
    - Where does it go?
      - The `WHERE` clause

Multi-Table SQL

```
Songs  (song_id,   song_name,   album_num,  weeks_in_top_40)

Artists(artist_id, artist_name, first_year_active)

Albums  (album_id,  album_name,  artist_num, year_released,
genre)
```

Example Query

```
Songs  (song_id,   song_name,   album_num,  weeks_in_top_40)

Artists(artist_id, artist_name, first_year_active)

Albums  (album_id,  album_name,  artist_num, year_released,
genre)
```

```sql
SELECT song_name, album_name
FROM Songs S, Artists A
WHERE S.album_num = A.album_num;
```

SQL Exercises

```
Songs  (song_id,   song_name,   album_num,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_num, year_released,
genre)
```

1. The name of all songs with the genre “country” which have spent more than 2 weeks in the top 40.

```
SELECT Songs.song_name
FROM Albums, Songs
WHERE Songs.album_num = Albums.album_id
     AND Albums.genre = 'country'
     AND Songs.weeks_in_top_40 > 2;
```

SQL Exercises

```
Songs  (song_id,   song_name,   album_num,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_num, year_released,
genre)
```

2. For each song, its name, the name of its album, and the name of its artist.

```
SELECT Songs.song_name, Albums.album_name, Artists.artist_name
FROM Artists, Albums, Songs
WHERE Artists.artist_id = Albums.artist_num
     AND Songs.album_num = Albums.album_id;
```

SQL Exercises

```
Songs  (song_id,   song_name,   album_num,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_num, year_released,
genre)
```

3. The number of albums released by each artist.

```
SELECT count(*)
FROM Artists, Albums
WHERE Artists.artist_id = Albums.artist_num
GROUP BY Artists.artist_id;
```

Join Algorithms

Cost Notation

- [R] = the number of pages required to store the relation
- |R| = the number of records in the relation
- PR = the number of records that fit on a single page of R

Simple Nested-Loops Join (SNLJ)

```
for record r in R:
 for record s in S:
  if theta(r, s):
   add join(r, s) to result
```

COST: IRI * [S] + [R]

Page Nested-Loops Join (PNLJ)

```
for page p_r in R:
 for page p_s in S:
  for record r in p_r:
   for record s in p_s:
    if theta(r, s):
     add join(r, s) to result
```

COST: [R] * [S] + [R]

Chunk Nested-Loops Join (CNLJ)

- This is very similar to PNLJ, but for one flaw win PNLJ.
  - PNLJ doesn't take advantage of all of memory.
- Instead of only loading *one* page of the outer relation into memory at a time, load B-2!

COST: ([R]/(B-2)) * [S] + [R]

Index Nested-Loops Join (INLJ)

```
for record r in R:
   for record s in S where r == s:
     add join(r, s) to result
```

This code is very similar to SNLJ, but the difference is that the second line does an index lookup.

Sort-Merge Join (SMJ)

```
Sort R on join attribute
Sort S on join attribute
Scan sorted-R and sorted-S in tandem to find matches
```

COST: cost(Sort(R)) + cost(Sort(S)) + [R] + [S]

Sort-Merge Join (SMJ)

Refinement:

Do the join during the final merging pass of sort.

Advantages of SMJ:

- one or both inputs is already sorted on join attribute
- output needs to be sorted on join attribute

COST: cost(Sort(R)) + cost(Sort(S)) + [R] + [S]

If you can sort in 2 passes: 3[R] + 3[S]

Hash Join

```
externally hash R on join attribute
externally hash S on join attribute
for partition p_r in r:
 build in memory hash table on a partition of R
 stream s and probe in memory hash table
```

COST: 3([R] + [S]) + [output] (for 2 passes)

Join Exercises

Give scenarios in which each of the following is optimal:

- CNLJ
- SMJ
- Hash join

Join Exercises

Give scenarios in which each of the following is optimal:

- CNLJ
  - Not using an equality predicate!
  - Join is just a cartesian product; very few distinct keys
- SMJ
  - Skewed input data (excepting worst cases)
  - Small memory size
  - Want sorted output/have sorted input
- Hash join
  - One partition is large, the other is small
  - Hybrid hashing

Join Exercises

We have 12 pages of memory, and we want to join two tables R and S where [R] is 100 pages and [S] is 50 pages.

1. How many disk *reads* are done to perform CNLJ?

2. What about hash join?

Join Exercises

We have 12 pages of memory, and we want to join two tables R and S where [R] is 100 pages and [S] is 50 pages.

1. How many disk *reads* are done to perform CNLJ?

   [S] + [S]/(B-2) * [R]

   50 + (50/10) * (100) = 550

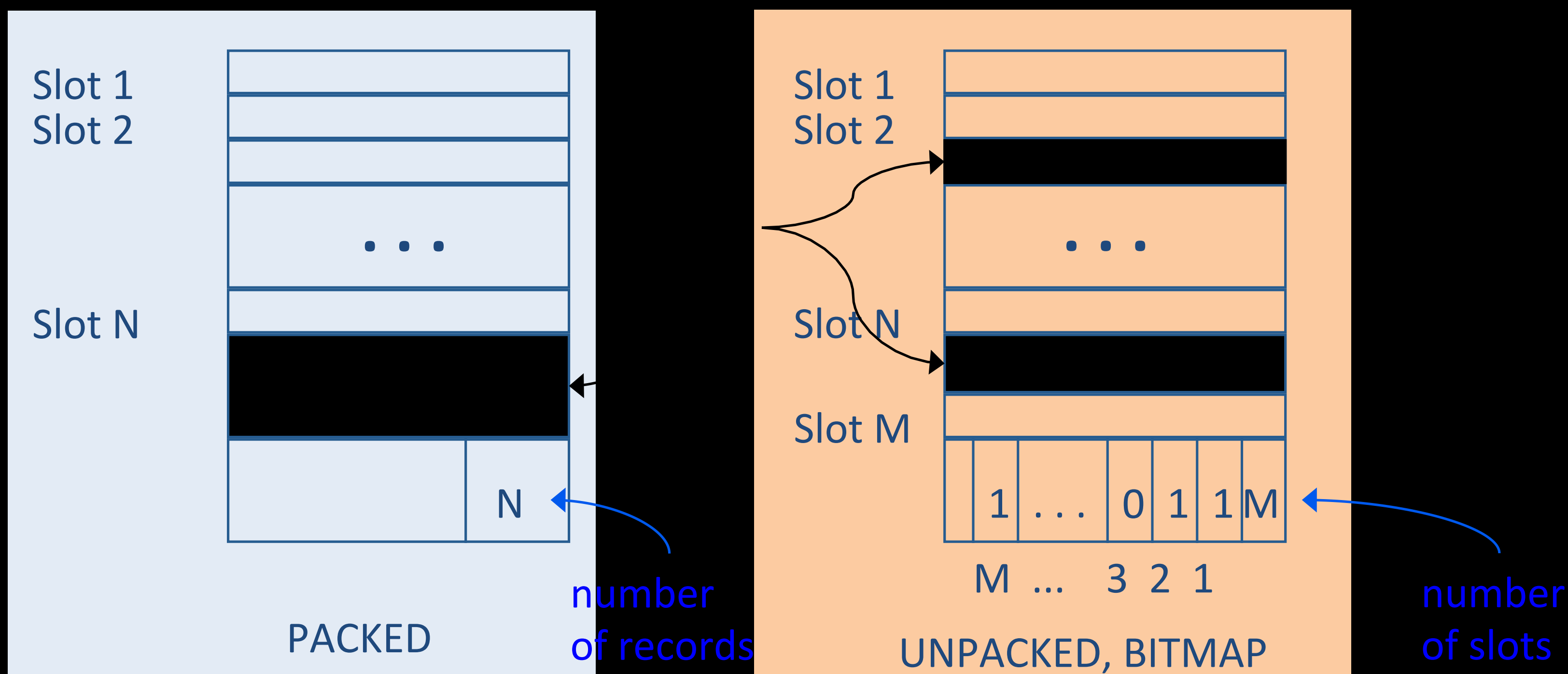
2. What about hash join?

   ([S] + [R]) * 2

   150 * 2 = 300

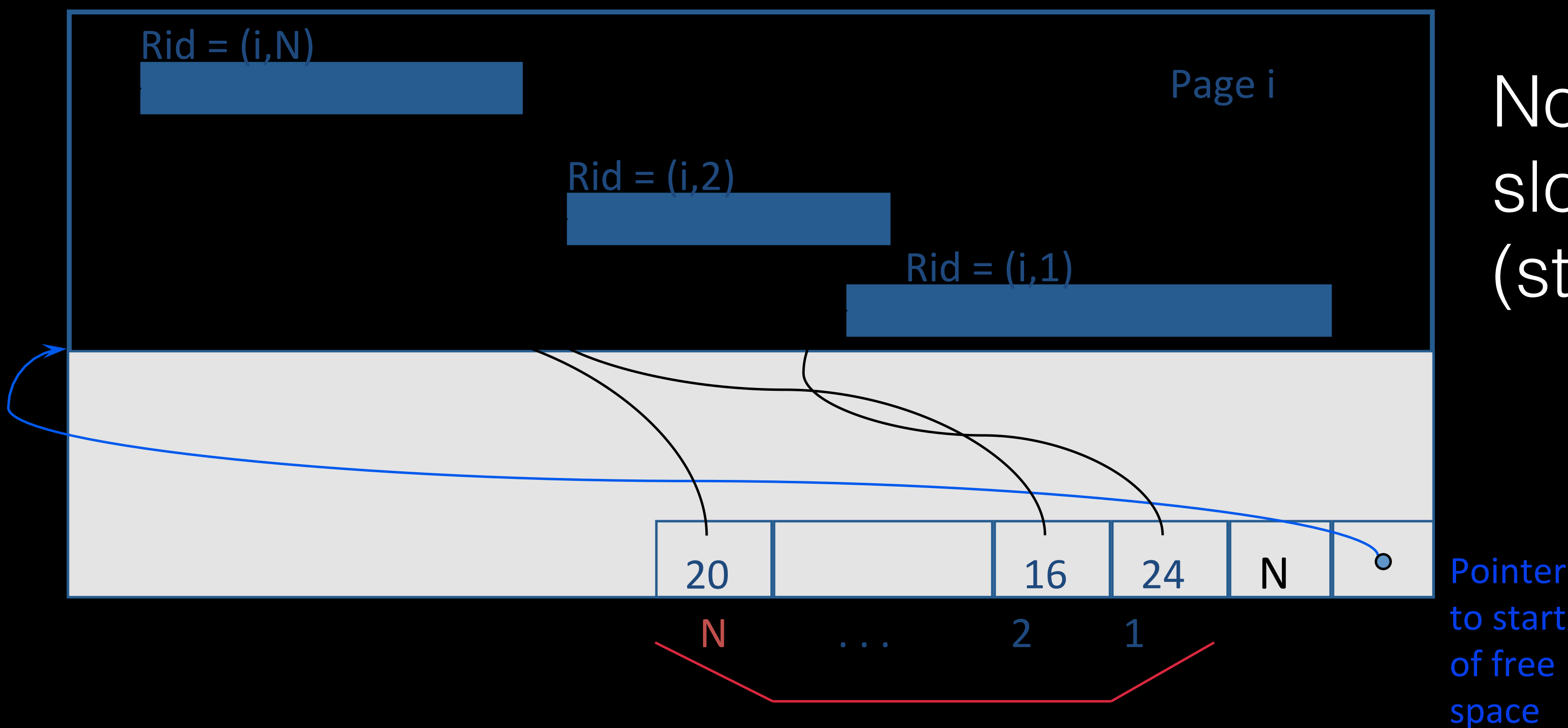
Files and Pages

Pages: Fixed-Length Records



<u>Record id</u> = <page id, slot #>

Pages: Variable Length Records (Slotted Pages)



Note: each slot entry is (start, length)

Pages & Files Exercises

What are the advantages and disadvantages of using slotted pages or bitmaps over just tightly packing records together?

Pages & Files Exercises

What are the advantages and disadvantages of using slotted pages or bitmaps over just tightly packing records together?

- allow movement of records without changing record ID
- slotted pages support variable-length records
- page directory incurs overhead - none necessary when using packed formats

Pages & Fiels Exercises

You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

What is the size of largest record you can insert?

How many 1-byte sized records can you insert?

Pages & Fiels Exercises

You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

What is the size of largest record you can insert?

A slot entry needs 4 bytes. (80 - 4) = 76 bytes.

How many 1-byte sized records can you insert?

Each record takes 1 byte for the record and 4 for the slot entry. That means we need 5 bytes per record. 80 bytes / 5 bytes = 16 records.