Introduction to Data Management CSE 344

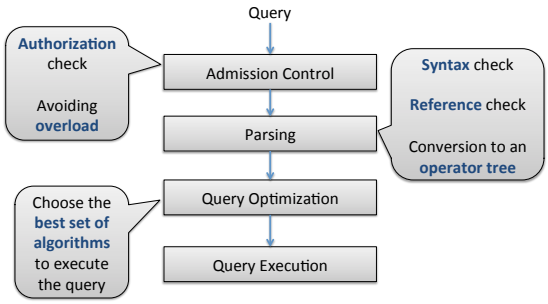
Extra Lecture: Basics of Query Evaluation

Presenter: Prasang Upadhyaya

Outline

- Life of a query in a DBMS
  - Query plans
- Indexing

Life of a Query



Query Plan

Output of parsing

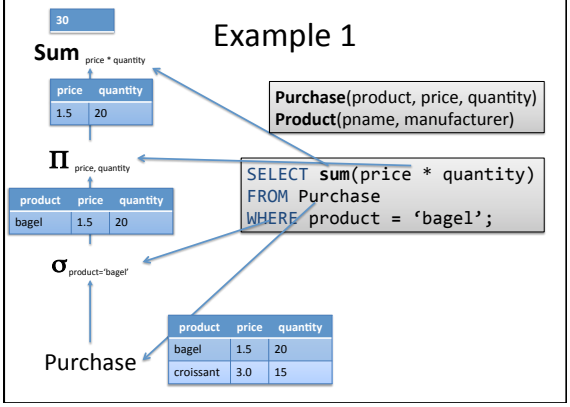
A **tree** of **relational operators**
Joins, Aggregates, Selections, Projections
Operators consume relations and produce relations

Leaves are **relations**

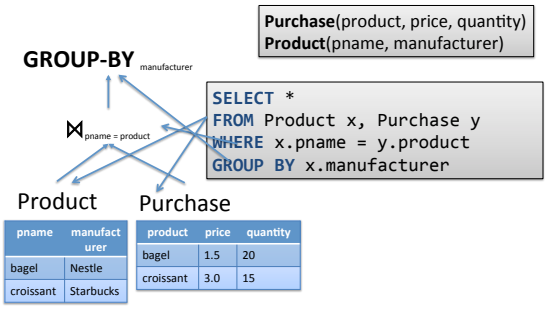
Output at the **root**

Every SQL statement is converted to a Query Plan

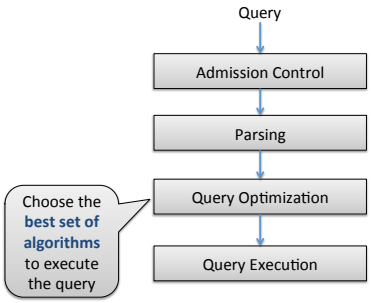
Example 1



Example 2



Life of a Query

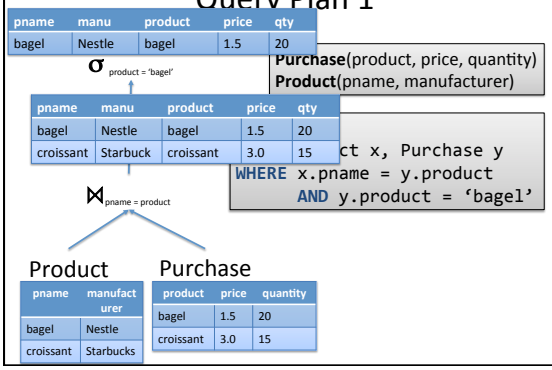


Query Optimization

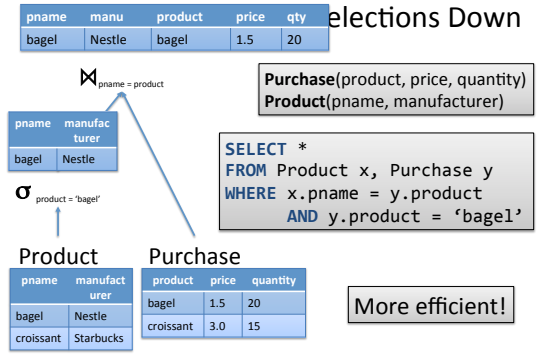
Find the **best way** to execute a query

Given a query plan
- Construct **alternate, equivalen**t plans
- Determine their **cost**
- Select the **cheapest** plan

Query Plan 1



elections Down

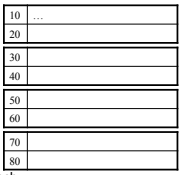


Cost of Query Plan

- Used to decide **which plan to pick**
- Cost approximates **how much time** the query would take
- Depends on:
  - How relations are **stored on disk**
  - How the **operator is implemented**

Data Storage

- DBMSs store data in **files**
- Most common organization is row-wise storage
- On disk, a file is split into blocks
- Each block contains a set of tuples



In the example, we have 4 blocks with 2 tuples each

Cost Parameters

- In database systems the data is on disk
- **Cost = total number of I/Os**
  - This is a simplification
  - Normally, need to consider IO, CPU, and network
- Cost of an operation = number of disk I/Os to
  - Read the operands
  - Compute the result
- Parameters:
  - **B(R) = # of blocks (i.e., pages) for relation R**
  - **T(R) = # of tuples in relation R**

Basic Join Algorithms

- Operator:
  - `Product` ⋈ `Company`
- Propose three algorithms for the join, assuming the tables fit in main memory:
  - **Hash join**
  - **Nested loop join**
  - **Sort-merge join**

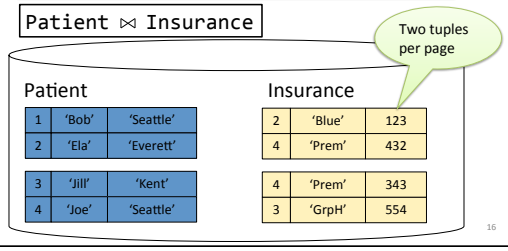
Hash Join

Hash join: R ⋈ S

- Scan R, build buckets in main memory
- Then scan S and join
- Cost: B(R) + B(S)

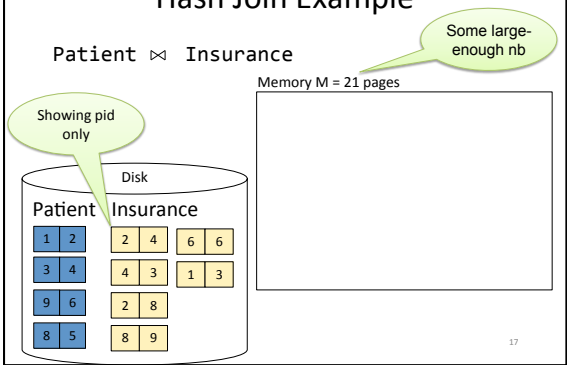
Hash Join Example

`Patient(pid, name, address)`
`Insurance(pid, provider, policy_nb)`
`Patient ⋈ Insurance`



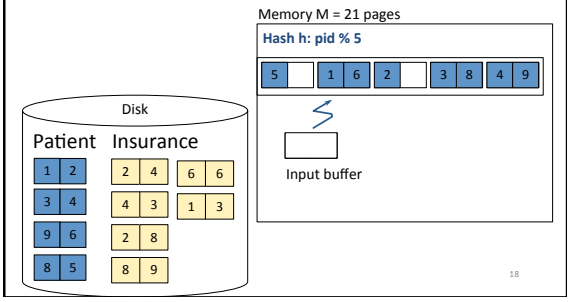
Hash Join Example

`Patient ⋈ Insurance`



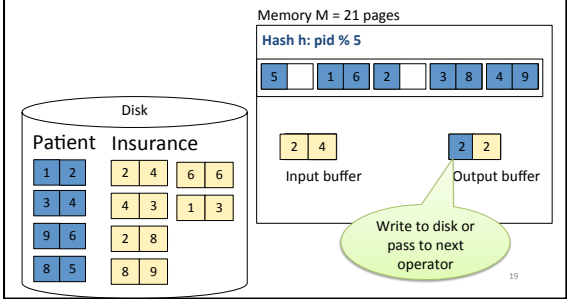
Hash Join Example

Step 1: Scan Patient and create hash table in memory



Hash Join Example

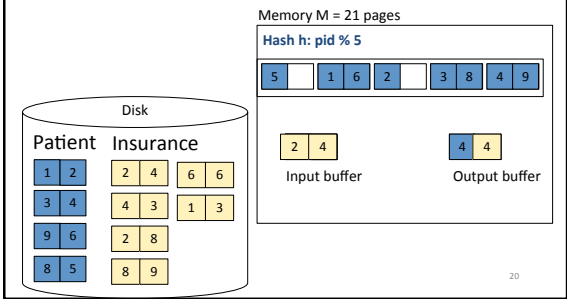
Step 2: Scan Insurance and probe into hash table



19

Hash Join Example

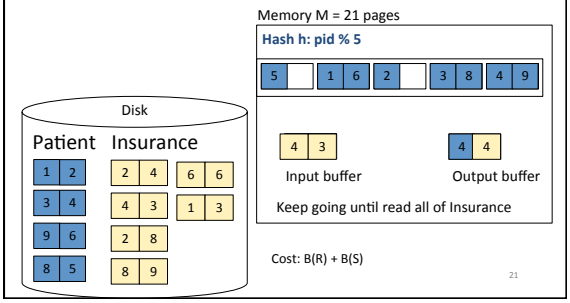
Step 2: Scan Insurance and probe into hash table



20

Hash Join Example

Step 2: Scan Insurance and probe into hash table



Cost: B(R) + B(S)

21

Nested Loop Joins

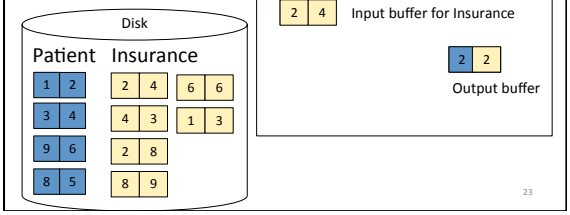
- Tuple-based nested loop R ⋈ S
- R is the outer relation, S is the inner relation

```
for each tuple r in R do
  for each tuple s in S do
    if r and s join then output (r,s)
```

- Cost: B(R) + B(R) B(S)

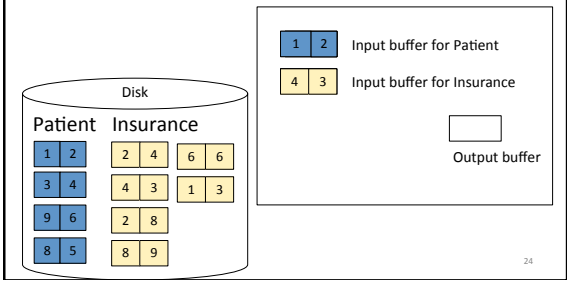
22

Nested Loop Example



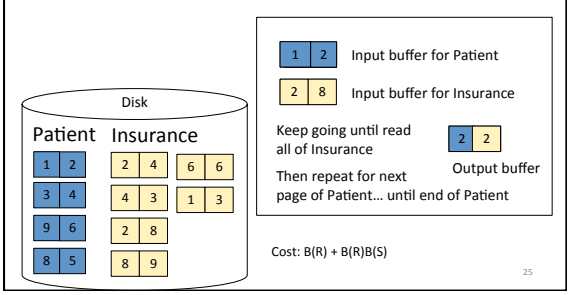
23

Nested Loop Example



24

Nested Loop Example



25

Sort-Merge Join

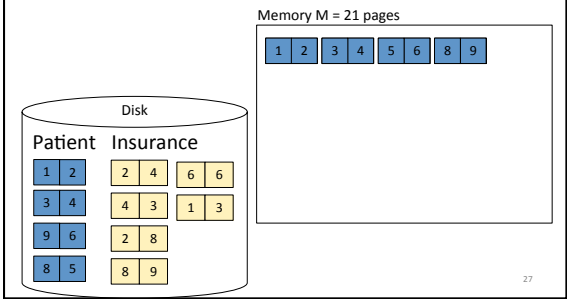
Sort-merge join: R ⋈ S

- Scan R and sort in main memory
- Scan S and sort in main memory
- Merge R and S

- Cost: B(R) + B(S)

Sort-Merge Join Example

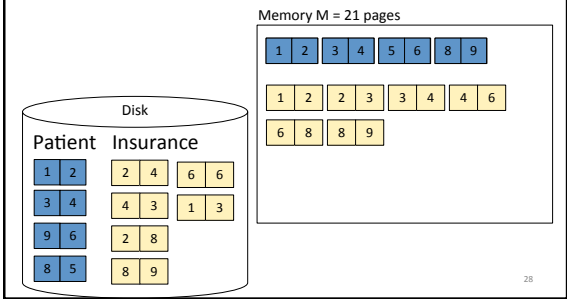
Step 1: Scan Patient and sort in memory



27

Sort-Merge Join Example

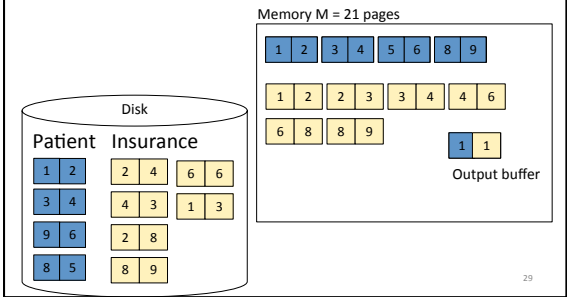
Step 2: Scan Insurance and sort in memory



28

Sort-Merge Join Example

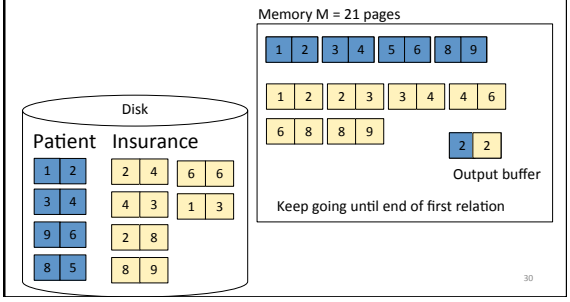
Step 3: Merge Patient and Insurance



29

Sort-Merge Join Example

Step 3: Merge Patient and Insurance



30

Indexes

- Special data structure for fast tuple look ups

Example

Patient

pid	name	address
1	Bob	Seattle
2	Ela	Everett
. . .		

Insurance

pid	provider	Policy_nb
2	Blue	123
. . .		

Both tables are on disk
How can we answer this query?

```
SELECT *
FROM  Patient p, Insurance i
WHERE p.pid=i.pid AND i.provider = 'Blue'
```


Possible Query Plan 1

```
For all patients
  For all insurance
    If conditions=true
      Return result
```

Nested-loop join

Relations may not fit in memory. So may need to read courses many times from disk

Possible Query Plan 2

Sort **patients** on **pid**
Sort **insurance** on **pid**
Merge join
Check additional conditions
Return **results that satisfy conditions**

Sort-merge join

Sorting can take a long time

Possible Query Plan 3

Find and **only read** from disk insurance from **provider 'Blue'**
For each such **policy**, find matching **patients**
Return **results**

Can we do this?

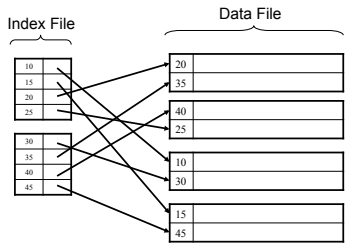
Yes! But we need **indexes**

Index

- An **additional** file, that allows fast access to records in the data file given a search key
- The index contains (key, value) pairs:
  - The key = an attribute value (e.g., pid or name)
  - The value = a pointer to the record

  "key" = "search key"
- SQL statement to create index:
  - `CREATE INDEX pid_index ON patient(pid)`

Example of Index



Selections with Indexing

```
SELECT *
FROM Insurance
WHERE provider = 'Blue'
```

Without index: Read the **whole** of Insurance.

With index: Read only those tuples that have 'Blue' as the insurance provider, if index on **provider** exists.

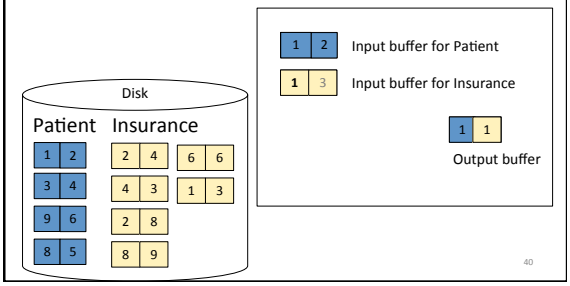
Nested Loop Joins

- Tuple-based nested loop R ⋈ S
- R is the outer relation, S is the inner relation

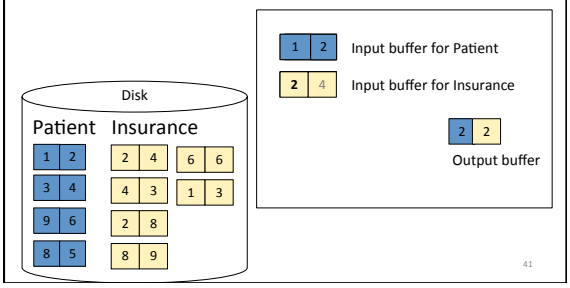
```
for each tuple r in R do
   use index on S to search for key
   then output (r,s)
```

- Cost: B(R) + B(R) * constant

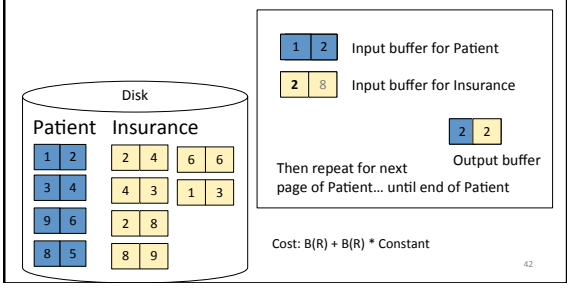
Nested Loop Example



Nested Loop Example



Nested Loop Example



Cost: B(R) + B(R) * Constant

Index Selection

- Depends on the query plan
- In general construct index on attributes that occur in
  - Selections
  - Join predicates
  - Grouping

Index Selection Example