ICS 624 Spring 2011
Overview of DB & IR

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

Example Relations

- Sailors(
  sid: integer,
  sname: string,
  rating: integer,
  age: real)
- Boats(
  bid: integer,
  bname: string,
  color: string)
- Reserves(
  sid: integer,
  bid: string,
  day: date)

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

B1

bid	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	green
104	Marine	Red

Basic SQL Query

```
SELECT [ DISTINCT ] target-list
FROM        relation-list
WHERE       qualification
```

- *relation-list* A list of relation names (possibly with a *range-variable* after each name).
- *target-list* A list of attributes of relations in *relation-list*
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of <, >, ≤, ≥, =, ≠) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated!

Example Q1

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND bid=103
```

Without range variables

```
SELECT sname
FROM   Sailors, Reserves
WHERE  Sailors.sid=Reserves.sid
       AND bid=103
```

- Range variables really needed only if the same relation appears twice in the FROM clause.
- Good style to always use range variables

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following *conceptual* evaluation strategy:
  1. Compute the cross-product of *relation-list.*
  2. Discard resulting tuples if they fail *qualifications*.
  3. Delete attributes that are not in *target-list.*
  4. If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

Example Q1: conceptual evaluation

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND bid=103
```

Conceptual Evaluation Steps:

1. Compute cross-product
2. Discard disqualified tuples
3. Delete unwanted attributes
4. If DISTINCT is specified, eliminate duplicate rows.

S.sid	sname	rating	age	R.sid	bid	day
22	Dustin	7	45	22	101	10/10/96
22	Dustin	7	45	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

S.sid	sname	rating	age	R.sid	bid	day
58	Rusty	10	35.0	58	103	11/12/96

sname
Rusty

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* (×) Allows us to combine two relations.
  - *Set-difference* (−) Tuples in reln. 1, but not in reln. 2.
  - *Union* (U) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

Projection

- Deletes attributes that are not in *projection list*.
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
- Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

$\pi_{sname, rating}(S2)$

sname	rating
Yuppy	9
Lubber	8
Guppy	5
Rusty	10

$\pi_{age}(S2)$

age
35.0
55.5
35.0
35.0

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

$\sigma_{\text{rating} > 8}\,(S2)$

sid	sname	rating	age
28	Yuppy	9	35.0
~~31~~	~~Lubber~~	~~8~~	~~55.5~~
~~44~~	~~Guppy~~	~~5~~	~~35.0~~
58	Rusty	10	35.0

$\pi_{\text{sname, rating}}\,(\sigma_{\text{rating}>8}\,(S2))$

~~sid~~	sname	rating	~~age~~
~~28~~	Yuppy	9	~~35.0~~
~~31~~	~~Lubber~~	~~8~~	~~55.5~~
~~44~~	~~Guppy~~	~~5~~	~~35.0~~
~~58~~	Rusty	10	~~35.0~~

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be union-compatible:
  - Same number of fields.
  - `Corresponding’ fields have the same type.
- What is the schema of result?

S1 U S2

sid	sname	rating	age
22	Dustin	7	45.0
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

S1

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S2

sid	sname	rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

Intersection & Set-Difference

S1 ∩ S2

sid	sname	rating	age
31	Lubber	8	55.5
58	Rusty	10	35.0

S1 − S2

sid	sname	rating	age
22	Dustin	7	45.0

S1

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S2

sid	sname	rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

Cross-Product

- Consider the cross product of S1 with R1
- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.
  - Rename to sid1 and sid2

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S1 × R1

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45	22	101	10/10/96
22	Dustin	7	45	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

Joins

- *Condition Join*: $R \bowtie_c S = \sigma_c(R \times S)$
- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

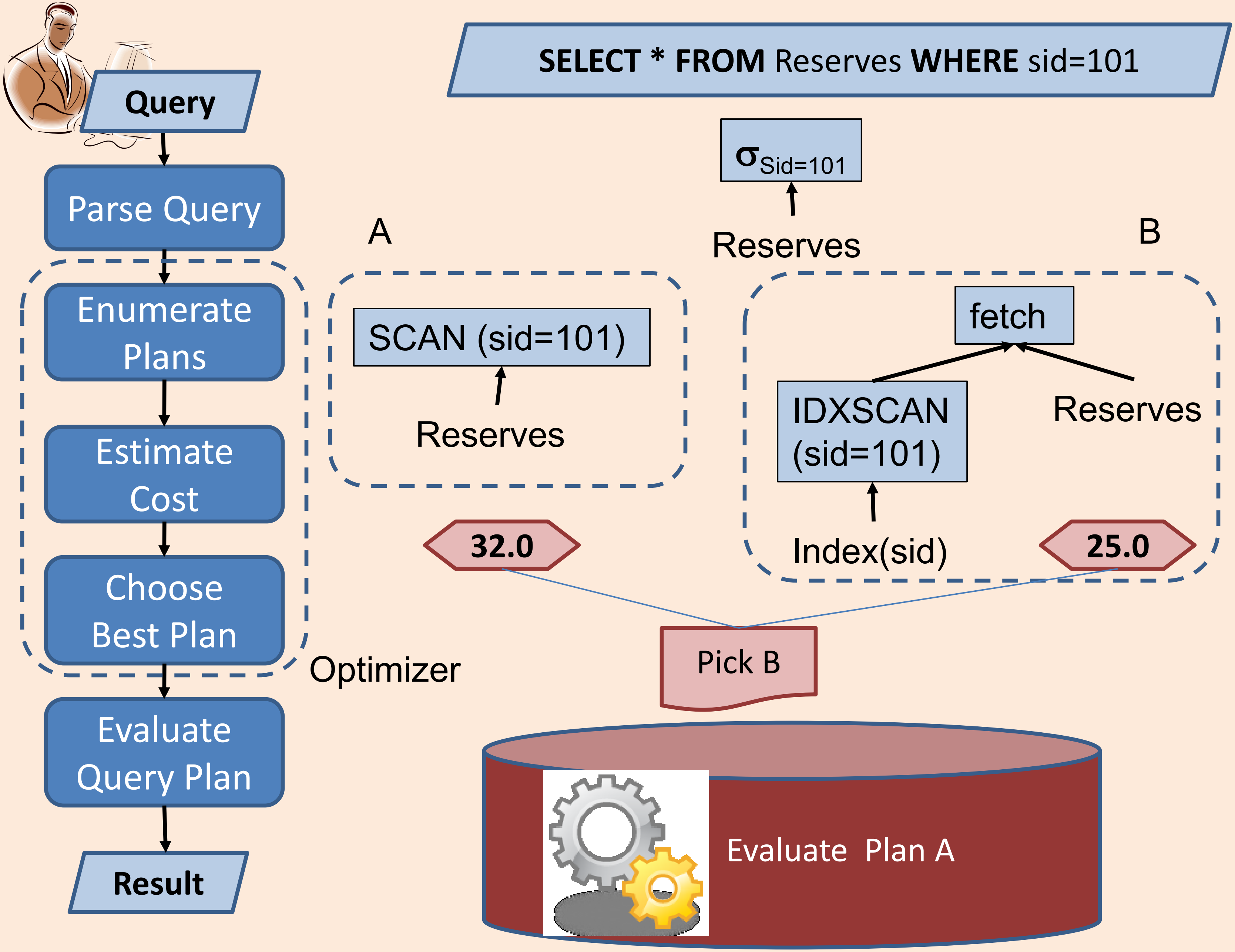
sid	sname	rating	age	sid	bid	day
22	Dustin	7	45	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

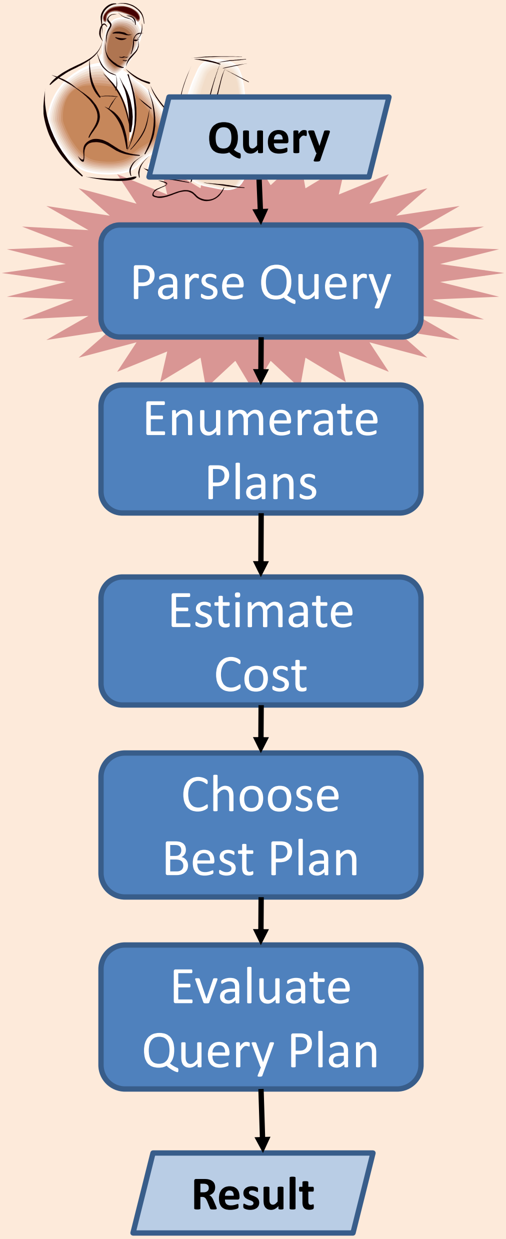
Equi-Joins & Natural Joins

- Equi-join: A special case of condition join where the condition c contains only *equalities*.
  - Result schema similar to cross-product, but only one copy of fields for which equality is specified.
- Natural Join: Equi-join on *all* common fields.

$$S1 \bowtie_{sid} R1$$

sid	sname	rating	age	bid	day
22	Dustin	7	45	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

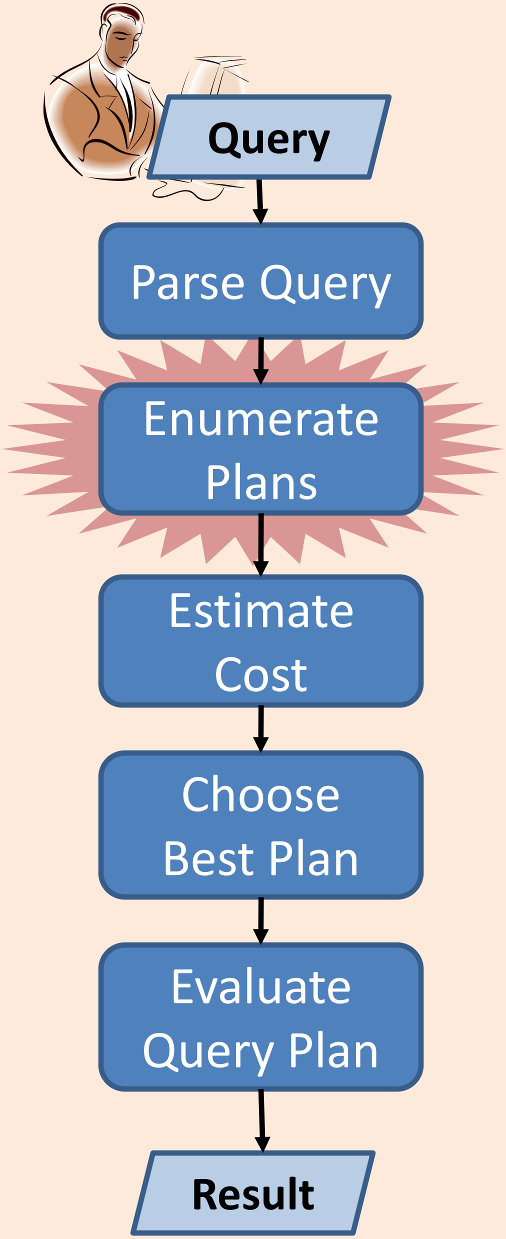

Query
Parse Query
Enumerate Plans
Estimate Cost
Choose Best Plan
Optimizer
Evaluate Query Plan
Result
SELECT * FROM Reserves WHERE sid=101
$\sigma_{Sid=101}$
Reserves
A
SCAN (sid=101)
Reserves
32.0
B
fetch
IDXSCAN (sid=101)
Reserves
Index(sid)
25.0
Pick B
Evaluate Plan A



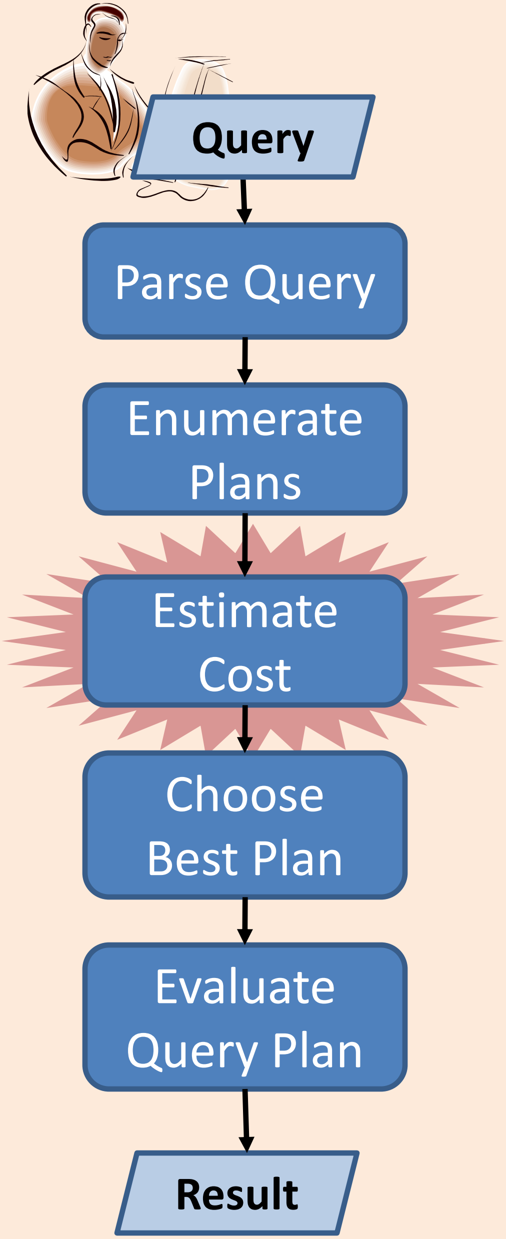
Parse Query

- Input : SQL
  - Eg. SELECT-FROM-WHERE, CREATE TABLE, DROP TABLE statements
- Output: Some data structure to represent the "query"
  - Relational algebra ?
- Also checks syntax, resolves aliases, binds names in SQL to objects in the catalog
- How ?

Enumerate Plans

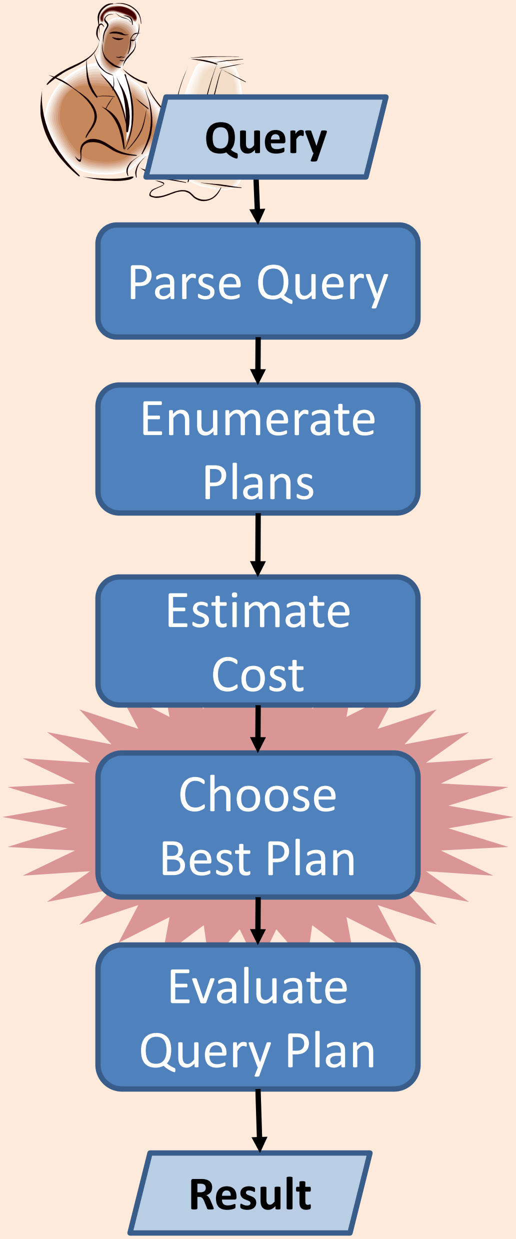


- **Input** : a data structure representing the "query"
- **Output**: a collection of equivalent query evaluation plans
- **Query Execution Plan** (QEP): tree of database operators.
  - high-level: RA operators are used
  - low-level: RA operators with particular implementation algorithm.
- **Plan enumeration**: find **equivalent** plans
  - Different QEPs that return the same results
  - Query rewriting : transformation of one QEP to another equivalent QEP.



Estimate Cost

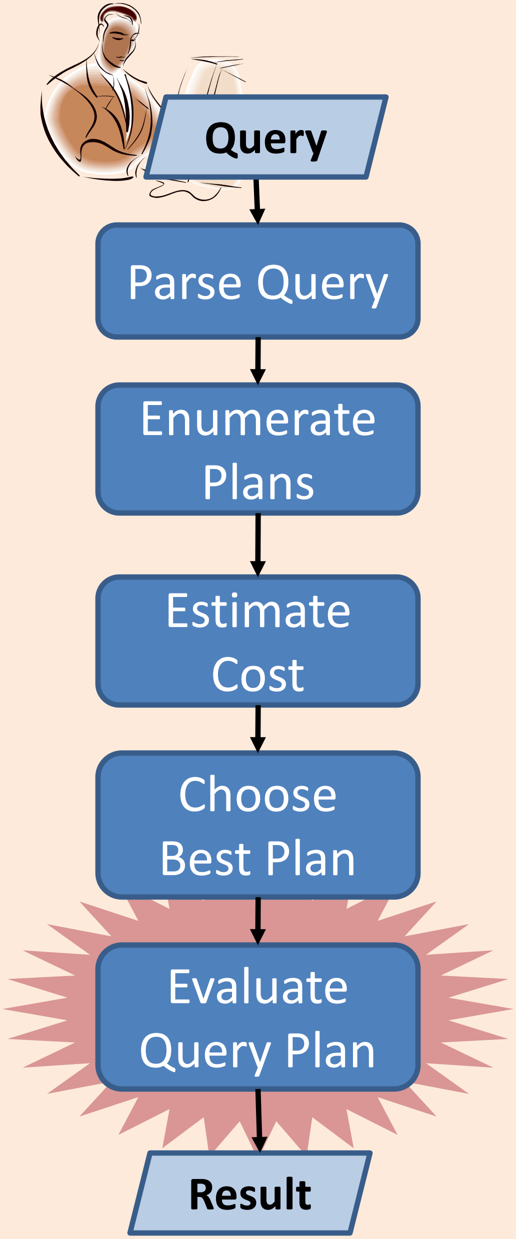
- **Input** : a collection of equivalent query evaluation plans
- **Output**: a cost estimate for each QEP in the collection
- **Cost estimation:** a mapping of a QEP to a cost
  - **Cost Model:** a model of what counts in the cost estimate. Eg. Disk accesses, CPU cost …
- Statistics about the data and the hardware are used.



Choose Best Plan

- **Input** : a collection of equivalent query evaluation plans and their cost estimate
- **Output**: best QEP in the collection
- The steps: enumerate plans, estimate cost, choose best plan collectively called the:
- **Query Optimizer:**
  - Explores the space of equivalent plan for a query
  - Chooses the best plan according to a cost model

Evaluate Query Plan



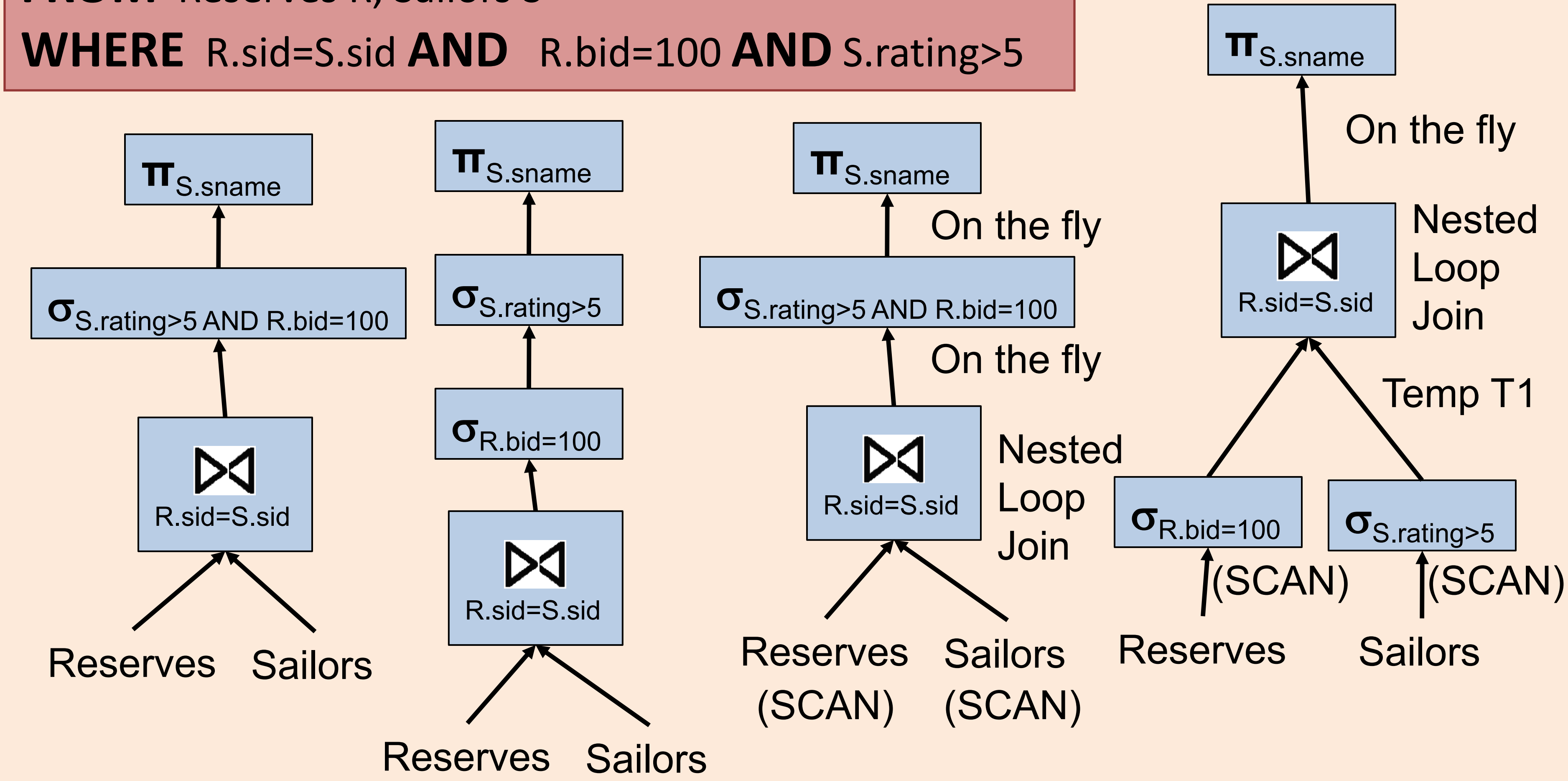
- **Input** : a QEP (hopefully the best)
- **Output**: Query results
- Often includes a "code generation" step to generate a lower level QEP in executable "code".
- **Query evaluation engine** is a "virtual machine" that executes some code representing low level QEP.

Query Execution Plans (QEPs)

- A **tree** of database operators: each operator is a RA operator with specific implementation
- **Selection σ**: Index Scan or Table Scan
- **Projection π**:
  - Without DISTINCT : Table Scan
  - With DISTINCT : requires sorting or index scan
- **Join** $\bowtie$:
  - Nested loop joins (naïve)
  - Index nested loop joins
  - Sort merge joins
- **Sort :**
  - In-memory sort
  - External sort

QEP Examples

SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid **AND** R.bid=100 **AND** S.rating>5



$\pi_{S.sname}$

$\sigma_{S.rating>5 \text{ AND } R.bid=100}$

$\bowtie$ R.sid=S.sid

Reserves Sailors

$\pi_{S.sname}$

$\sigma_{S.rating>5}$

$\sigma_{R.bid=100}$

$\bowtie$ R.sid=S.sid

Reserves Sailors

$\pi_{S.sname}$

On the fly

$\sigma_{S.rating>5 \text{ AND } R.bid=100}$

On the fly

$\bowtie$ R.sid=S.sid

Nested Loop Join

Reserves (SCAN) Sailors (SCAN)

$\pi_{S.sname}$

On the fly

$\bowtie$ R.sid=S.sid

Nested Loop Join

Temp T1

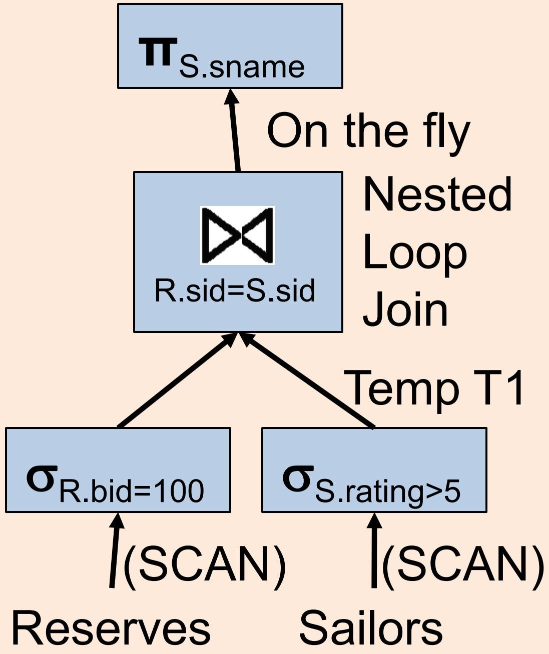
$\sigma_{R.bid=100}$ $\sigma_{S.rating>5}$

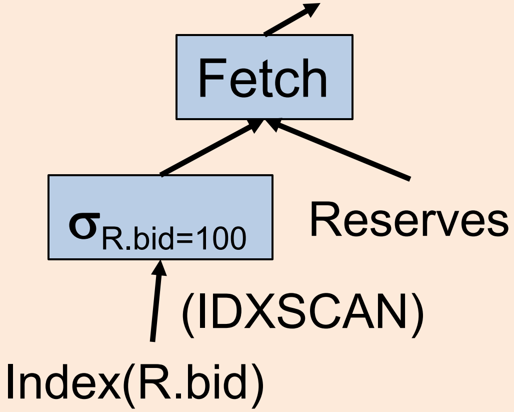
(SCAN) (SCAN)

Reserves Sailors

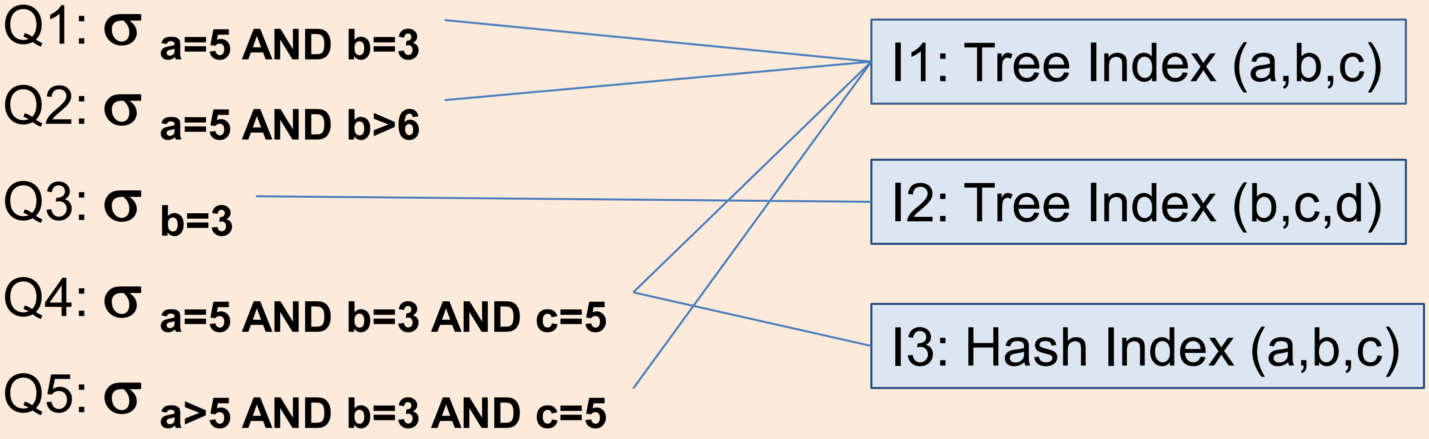
Access Paths

- An **access path** is a method of retrieving tuples. Eg. Given a query with a selection condition:
  - File or table scan
  - Index scan
- **Index matching problem:** given a selection condition, which indexes can be used for the selection, i.e., matches the selection ?
  - Selection condition normalized to conjunctive normal form (CNF), where each term is a *conjunct*
  - Eg. (day<8/9/94 **AND** rname=‘Paul’) **OR** bid=5 **OR** sid=3
  - **CNF**: (day<8/9/94 **OR** bid=5 **OR** sid=3) AND (rname=‘Paul’ **OR** bid=5 **OR** sid=3)





Index Matching



Q1: $\sigma_{a=5 \text{ AND } b=3}$

Q2: $\sigma_{a=5 \text{ AND } b>6}$

Q3: $\sigma_{b=3}$

Q4: $\sigma_{a=5 \text{ AND } b=3 \text{ AND } c=5}$

Q5: $\sigma_{a>5 \text{ AND } b=3 \text{ AND } c=5}$

I1: Tree Index (a,b,c)

I2: Tree Index (b,c,d)

I3: Hash Index (a,b,c)

- A <u>**tree index**</u> matches a selection condition if the selection condition is a prefix of the index search key.
- A <u>**hash index**</u> matches a selection condition if the selection condition has a term *attribute=value* for every attribute in the index search key

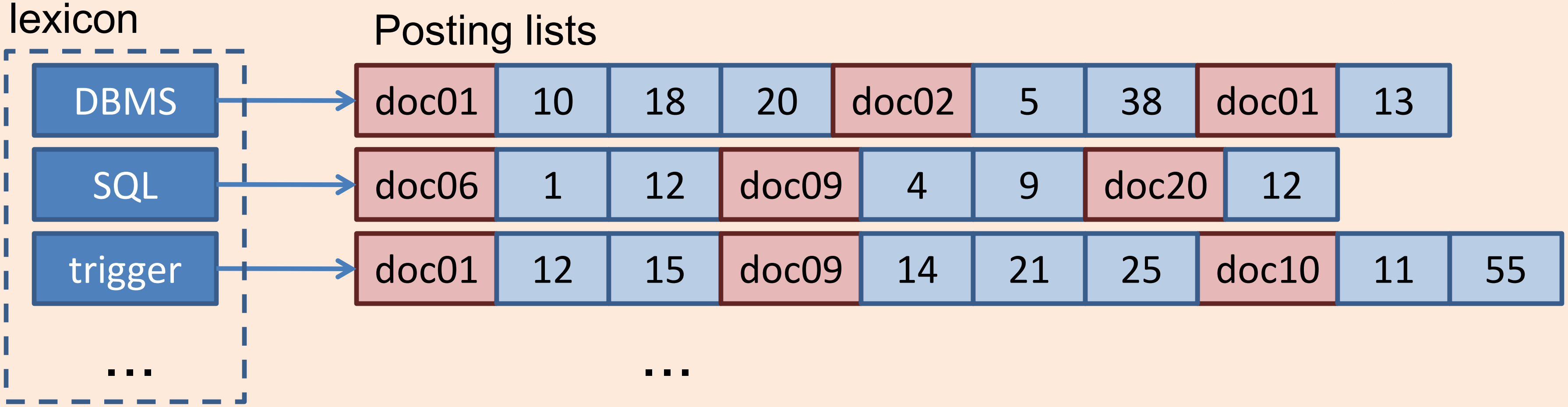
Unstructured Text Data

- Field of “Information Retrieval”
- Data Model
  - Collection of documents
  - Each document is a bag of words (aka terms)
- Query Model
  - Keyword + Boolean Combinations
  - Eg. DBMS and SQL and tutorial
- Details:
  - Not all words are equal. “Stop words” (eg. “the”, “a”, “his” ...) are ignored.
  - Stemming : convert words to their basic form. Eg. “Surfing”, “surfed” becomes “surf”

Inverted Indexes

- Recall: an index is a mapping of search key to data entries
  - What is the search key ?
  - What is the data entry ?
- Inverted Index:
  - For each term store a list of postings
  - A posting consists of <docid,position> pairs

What is the data in an inverted index sorted on ?



lexicon

Posting lists

DBMS → doc01 10 18 20 doc02 5 38 doc01 13

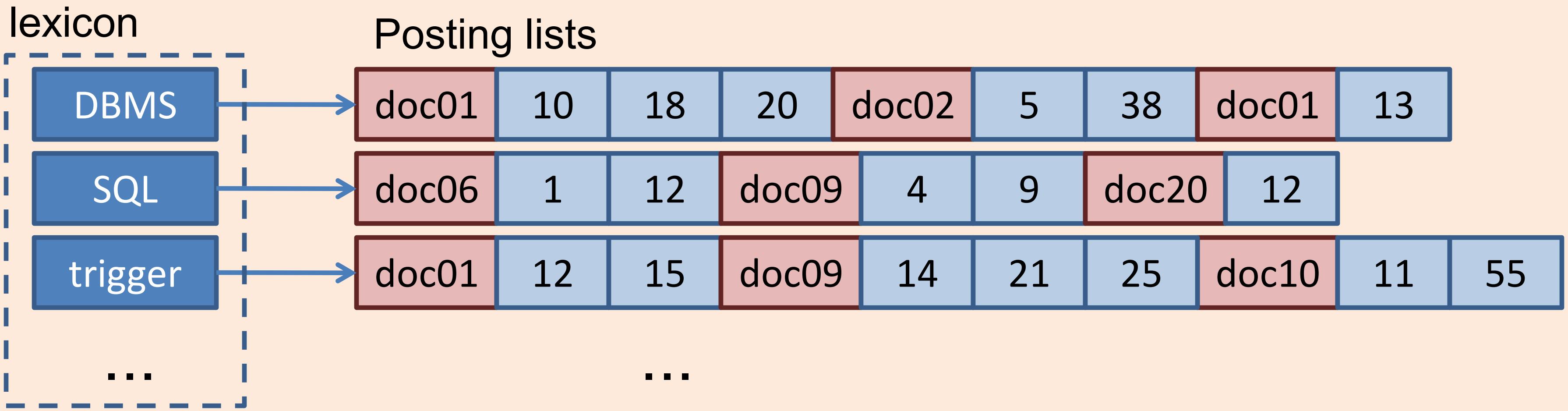
SQL → doc06 1 12 doc09 4 9 doc20 12

trigger → doc01 12 15 doc09 14 21 25 doc10 11 55

…

…

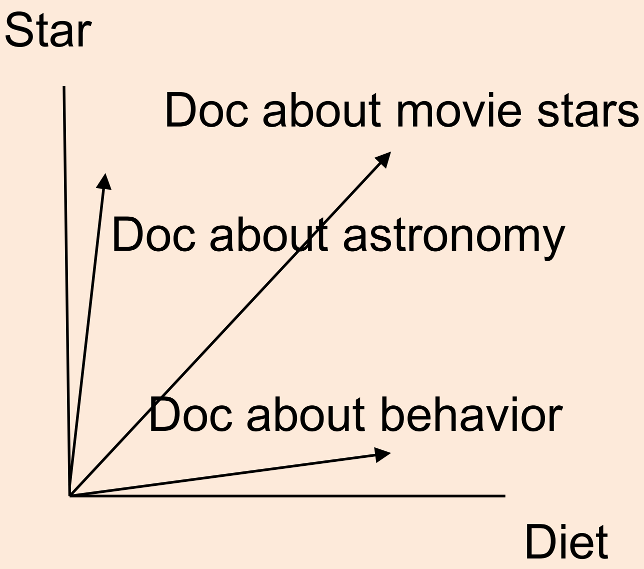
Lookups using Inverted Indexes


lexicon
Posting lists
DBMS
doc01 10 18 20 doc02 5 38 doc01 13
SQL
doc06 1 12 doc09 4 9 doc20 12
trigger
doc01 12 15 doc09 14 21 25 doc10 11 55
…
…

- Given a single keyword query "k" (eg. SQL)
  - Find k in the lexicon
  - Retrieve the posting list for k
  - Scan posting list for document IDs [and positions]
- What if the query is "k1 and k2" ?
  - Retrieve document IDs for k1 and k2
  - Perform intersection

Too Many Matching Documents

- Rank the results by “relevance”!
- Vector-Space Model
  - Documents are vectors in hi-dimensional space
  - Each dimension in the vector represents a term
  - Queries are represented as vectors similarly
  - Vector distance (dot product) between query vector and document vector gives ranking criteria
  - Weights can be used to tweak relevance
- PageRank (later)



How good are the retrieved docs?

- *Precision* : Fraction of retrieved docs that are relevant to user’s information need
- *Recall* : Fraction of relevant docs in collection that are retrieved