ICS 624 Spring 2013
Overview of Database Systems

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa

Outline

- The Data Management Problem
- Data, Databases and DBMSs
- The Relational Model
- Transactions
- Structured Query Language
- Relational Algebra
- Query Processing in DBMSs

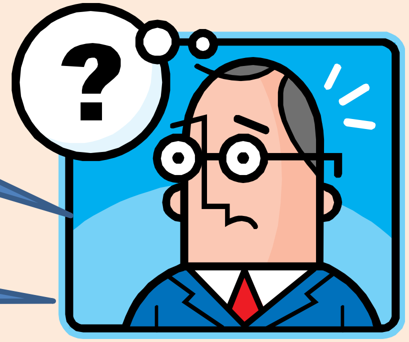
The Data Management Problem

Where is the photo I took last Christmas ?

Where did I read about "Turing Machines" ?

Where is the invoice for this computer ?

Which product is the most profitable ?



User

Queries

?

Data

What is ``data'' ?

- **Data** are known facts that can be recorded and that have implicit meaning.
- Three broad categories of data
  - Structured data
  - Semi-structured data
  - Unstructured data
- ``Structure'' of data refers to the organization within the data that is identifiable.

What is a database ?

- A **database** : a collection of related data.
  - Represents some aspect of the real world (aka universe of discourse).
  - Logically coherent collection of data
  - Designed and built for specific purpose
- A **data model** is a collection of concepts for describing/organizing the data.
- A **schema** is a description of a particular collection of data, using the a given data model.

The Relational Data Model

- *Relational database:* a set of *relations*
- A *relation* is made up of 2 parts:
  - *Instance* : a *table,* with rows and columns. #Rows = *cardinality,* #fields = *degree / arity.*
  - *Schema* : specifies name of relation, plus name and domain/type of each column or attribute.
    - E.G. Students(sid: string, name: string, login: string, age: integer, gpa: real).
- Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).

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

Why is the relational model useful ?

- Supports simple and powerful query capabilities!
- Structured Query Language (SQL)

```
SELECT S.sname
FROM  Students S
WHERE  S.gpa>3.5
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

What is a DBMS ?

- A **database management system (DBMS)** is a *collection of programs* that enables users to
  - Create new DBs and specify the structure using data definition language (DDL)
  - Query data using a query language or data manipulation language (DML)
  - Store very large amounts of data
  - Support durability in the face of failures, errors, misuse
  - Control concurrent access to data from many users

Types of Databases

- On-line Transaction Processing (OLTP)
  - Banking
  - Airline reservations
  - Corporate records
- On-line Analytical Processing (OLAP)
  - Data warehouses, data marts
  - Business intelligence (BI)
- Specialized databases
  - Multimedia
  - XML
  - Geographical Information Systems (GIS)
  - Real-time databases (telecom industry)
- Special Applications
  - Customer Relationship Management (CRM)
  - Enterprise Resource Planning (ERP)
- Hosted DB Services
  - Amazon, Salesforce

Transactions

- A *transaction* is the DBMS's abstract view of a user program: a sequence of reads and writes.
  - Eg. User 1 views available seats and reserves seat 22A.
- A DBMS supports multiple users, ie, multiple transactions may be running concurrently.
  - Eg. User 2 views available seats and reserves seat 22A.
  - Eg. User 3 views available seats and reserves seat 23D.

ACID Properties of Transactions

- **A**tomicity : all-or-nothing execution of transactions
- **C**onsistency: constraints on data elements is preserved
- **I**solation: each transaction executes as if no other transaction is executing concurrently
- **D**urability: effect of an executed transaction must never be lost

A Bit of History

- 1970 Edgar F Codd (aka "Ted") invented the relational model in the seminal paper "A Relational Model of Data for Large Shared Data Banks"
  - Main concept: *relation* = a table with rows and columns.
  - Every relation has a *schema*, which describes the columns.
- Prior 1970, no standard data model.
  - Network model used by Codasyl
  - Hierarchical model used by IMS
- After 1970, IBM built System R as proof-of-concept for relational model and used **SQL** as the query language. SQL eventually became a standard.

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 $<, >, \leq, \geq, =, \neq$) 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
```

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

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
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* (∪) 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.*)

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

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

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

~~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

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

S1

<u>sid</u>	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*.

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

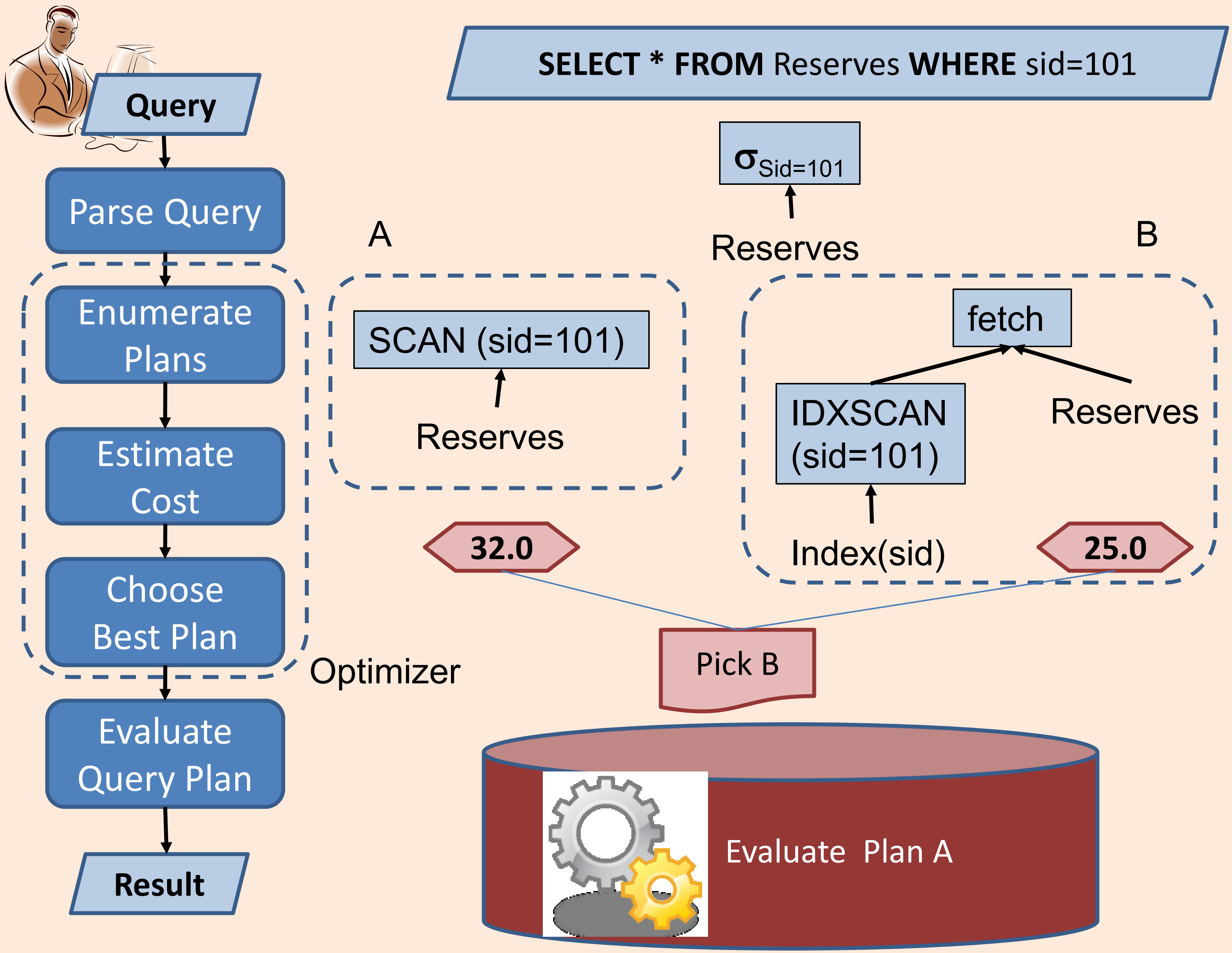
sid	sname	rating	age	sid	bid	day
22	Dustin	7	45	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

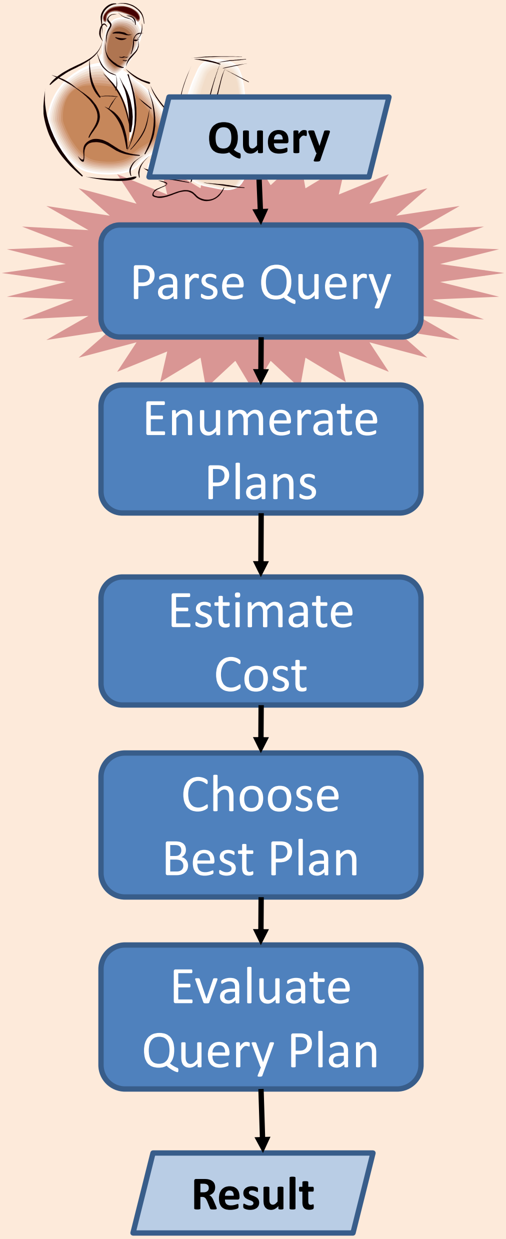
Equi-Joins & Natural Joins

- 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.

$$S1 \bowtie_{sid} R1$$

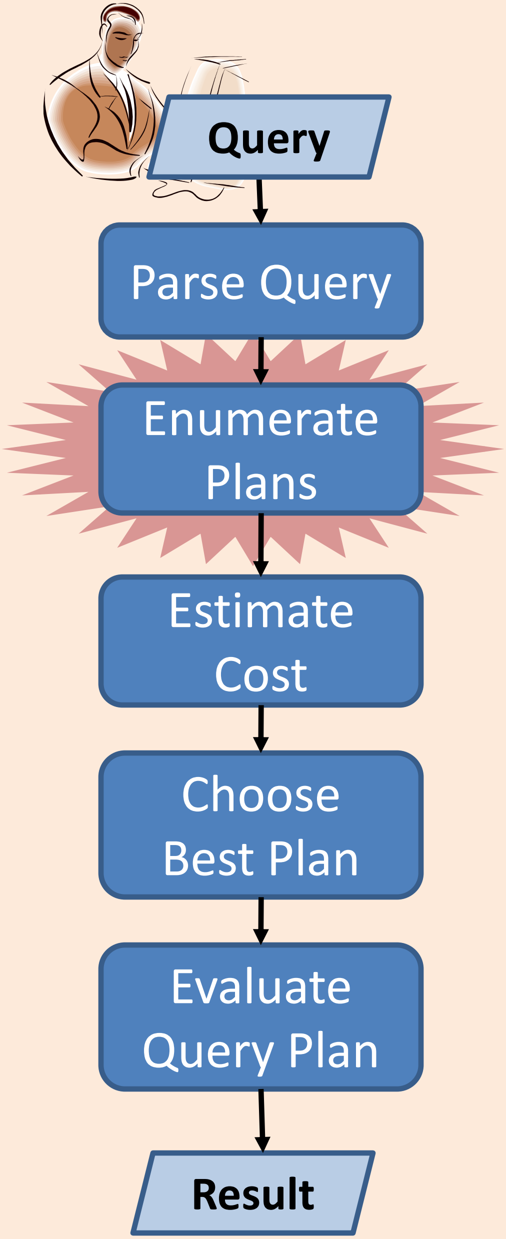
sid	sname	rating	age	bid	day
22	Dustin	7	45	101	10/10/96
58	Rusty	10	35.0	103	11/12/96


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



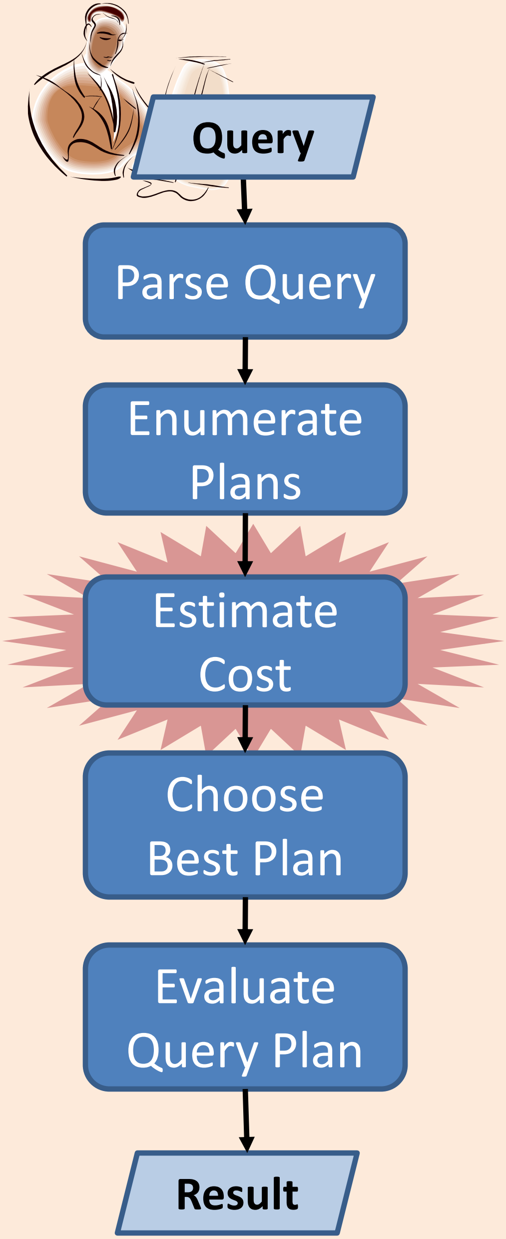
Parse Query

- 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 ?



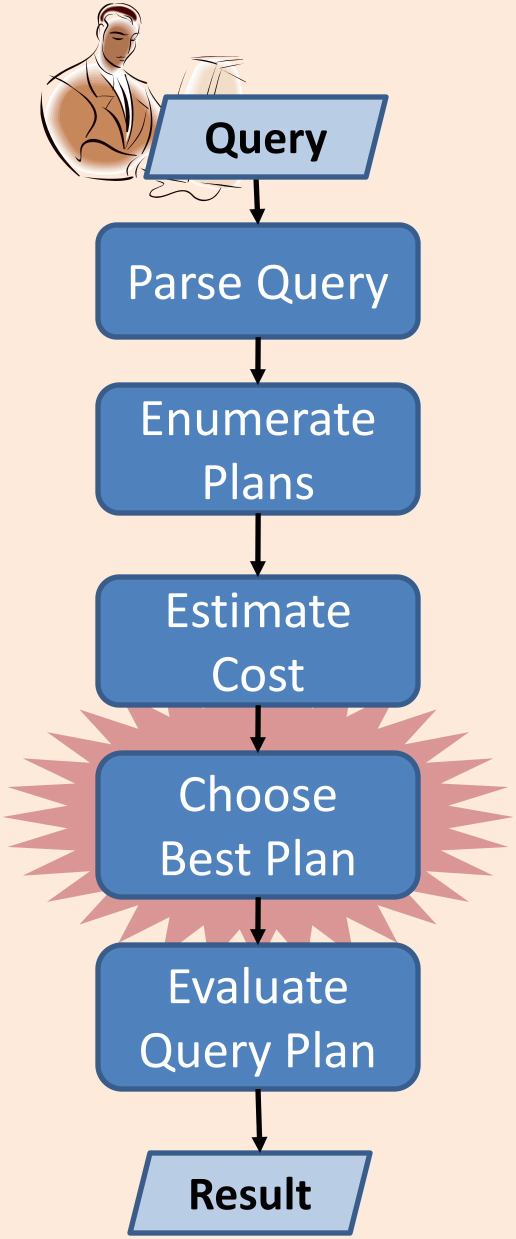
Enumerate Plans

- **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.



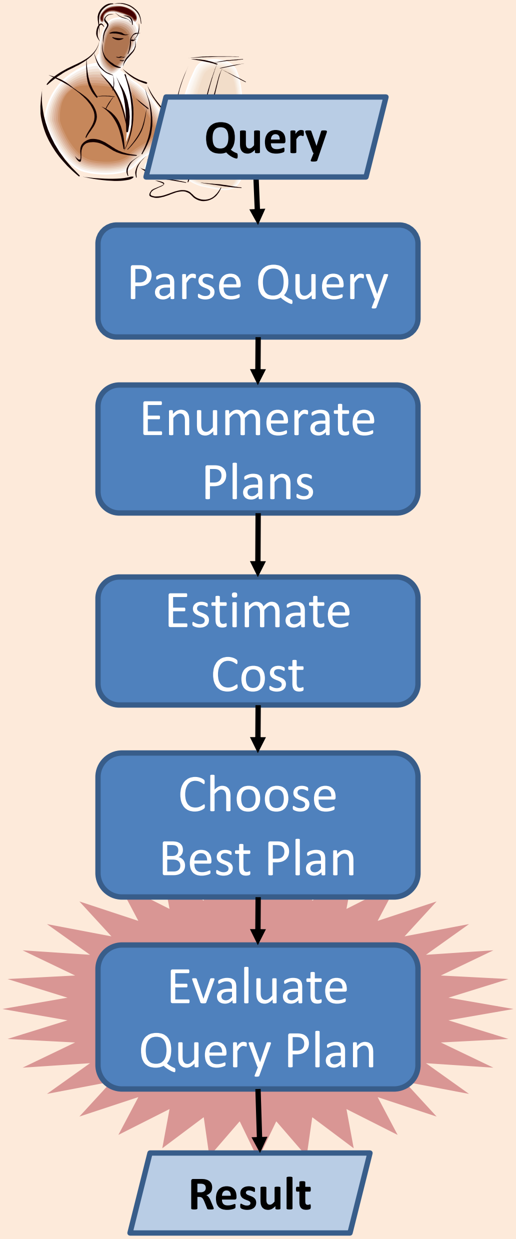
Estimate Cost

- **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

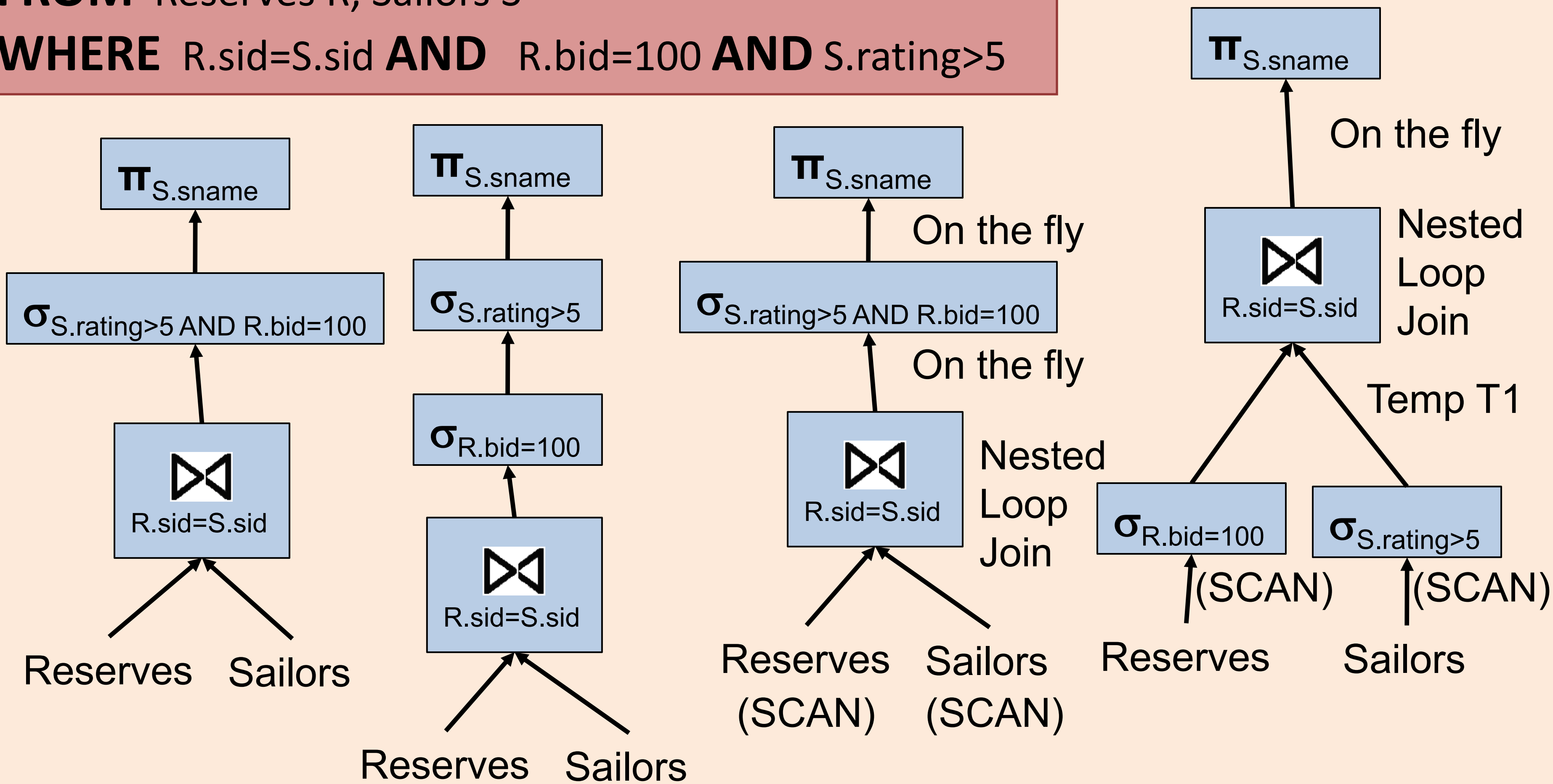
- **Input** : a QEP (hopefully the best)
- **Output**: Query results
- Often includes a "code generation" step to generate a lower level QEP in executable "code".
- **Query evaluation engine** is a "virtual machine" that executes some code representing low level QEP.

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

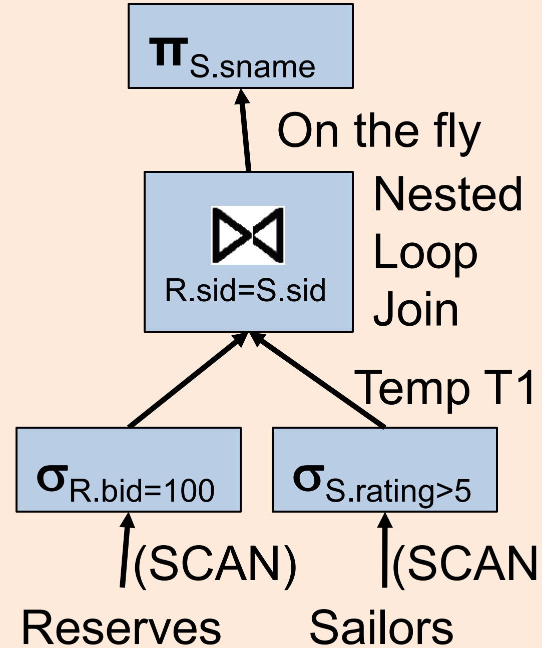
QEP Examples

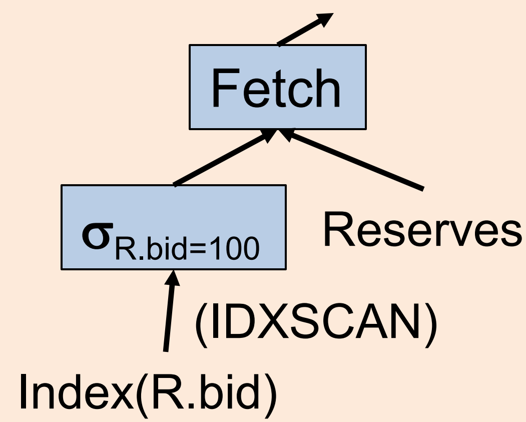
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid **AND** R.bid=100 **AND** S.rating>5



Access Paths

- An **<u>access path</u>** 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