Queries

- Q: Find ID and text of all page written by Bob and containing the word «db»

Step2:

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

users

id	name	karma
729	Bob	35
730	John	0

posts

id	text	ts	authorId
33981	'Hello DB!'	1493897351	729
33982	'Show me code'	1493904323	812

How is a Query Answered?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	'Hello DB!'	...	729	729	Bob	35
33981	'Hello DB!'	...	729	730	John	0
33982	'Show me code'	...	812	729	Bob	35
33982	'Show me code'	...	812	730	John	0

p

id	text	ts	authorId
33981	'Hello DB!'	...	729
33982	'Show me code'	...	812

u

id	name	karma
729	Bob	35
730	John	0

How is a Query Answered?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

where(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	'Hello DB!'	...	729	729	Bob	35

(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	'Hello DB!'	...	729	729	Bob	35
33981	'Hello DB!'	...	729	730	John	0
33982	'Show me code'	...	812	729	Bob	35
33982	'Show me code'	...	812	730	John	0

How is a Query Answered?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

select(where(p, u))

p.id	p.text
33981	'Hello DB!'

where(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	'Hello DB!'	...	729	729	Bob	35

Some issues

- To retrieve all data in a table, asterisk (*) is used; however, it is not a good practice to use the asterisk (*) in the SELECT statement if you have a big table with many columns and many rows:
  - Much of the retrieved data may not be necessary for you.
  - Retrieving unnecessary data from a big table degrades the performance of the database server and your application.

Why do we need it to be fast?

SQL and Relational Algebra

Review on Relational Algebra (CH5)

- Relational algebra and relational calculus are formal languages associated with the relational model.
- Informally, relational algebra is a (high-level) procedural language and relational calculus a non-procedural language.
- However, formally both are equivalent to one another.
- A language that produces a relation that can be derived using relational calculus is **relationally complete**.

Relational Algebra (1 of 2)

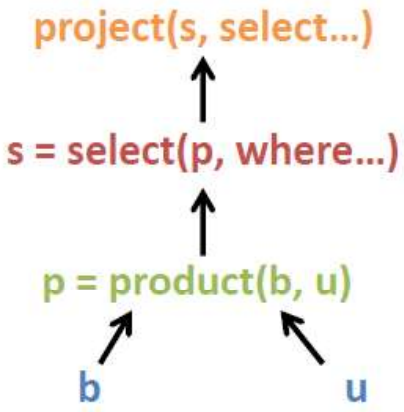
- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Both operands and results are relations, so output from one operation can become input to another operation.
- Allows expressions to be nested, just as in arithmetic. This property is called **closure**.

Why this translation?

SQL and Relational Algebra

- SQL is difficult to implement directly
  - A single SQL command can embody several tasks
- Relational algebra is relatively easy to implement
  - Each ***operator*** denotes a small, well-defined task

```
SELECT b.blog_id
    FROM blog_pages b, users u
    WHERE b.author_id=u.user_id
      AND u.name='Steven Sinofsky'
      AND b.created >= 2011/1/1;
```



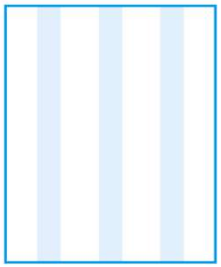
Relational Algebra (2 of 2)

- Five basic operations in relational algebra:
- *Selection* (σ) Selects a subset of *rows* from relation (horizontal).
- *Projection* (π) Retains only wanted *columns* from relation (vertical).
- *Cross-product* (×) Allows us to combine two relations.
- *Set-difference* (—) Tuples in r1, but not in r2.
- *Union* (∪) Tuples in r1 and/or in r2.
- *Intersection* (∩)
- *Join* (⋈)
- *Division* (/)

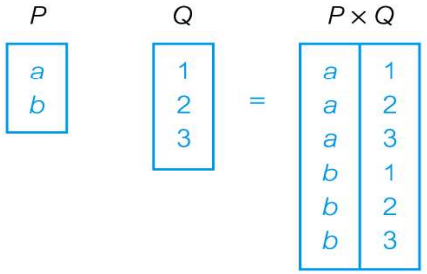
- These perform most of the data retrieval operations needed.

Relational Algebra Operations (1 of 2)

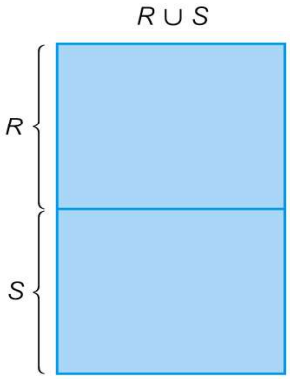
(a) Selection



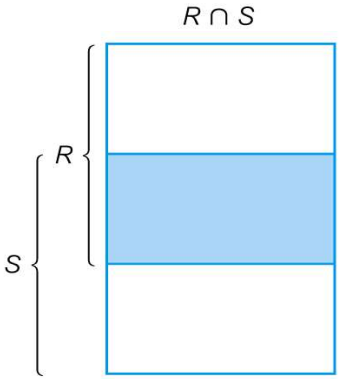
(b) Projection



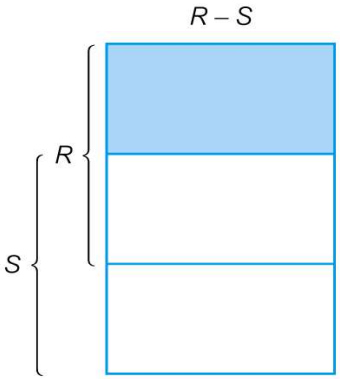
(c) Cartesian product



(d) Union

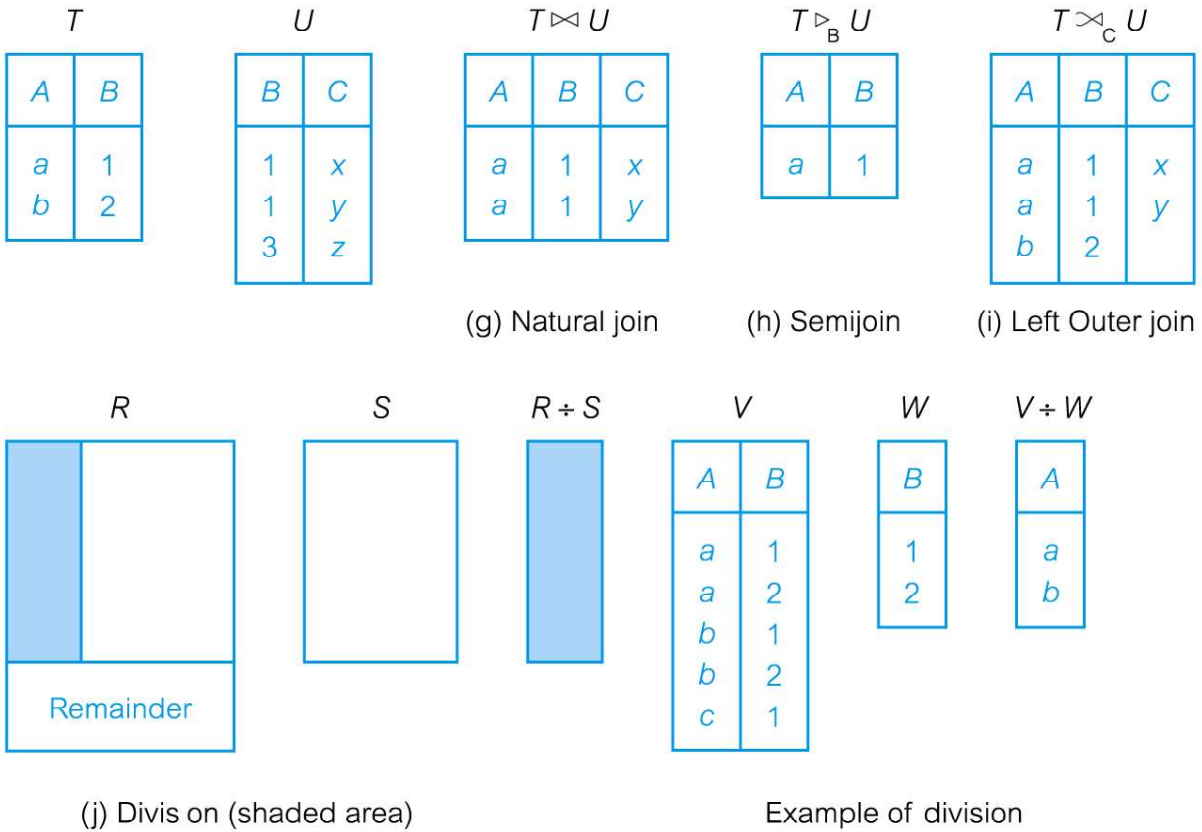


(e) Intersection



(f) Set difference

Relational Algebra Operations (2 of 2)



Example - Selection (or Restriction)

- List all staff with a salary greater than £10,000.

$$\sigma_{salary>10000}(Staff)$$

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

– Works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (**predicate**).

Example - Projection

- Produce a list of salaries for all staff, showing only staffNo, f Name, lName, and salary details.

$$\Pi_{\text{staffNo, fName, lName, salary}} (\text{Staff})$$

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

Works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.

Example - Union

- List all cities where there is either a branch office or a property for rent.

$$\Pi_{city}(Branch) \cup \Pi_{city}(PropertyForRent)$$

city
London
Aberdeen
Glasgow
Bristol

Union compatible:
1) Same number of fields
2) Same domain

Example - Set Difference

- List all cities where there is a branch office but no properties for rent.

$$\Pi_{city}(Branch) - \Pi_{city}(PropertyForRent)$$

city
Bristol

Defines a relation consisting of the tuples that are in relation R, but not in S.

Example - Intersection

- List all cities where there is both a branch office and at least one property for rent.

$$\Pi_{city}(Branch) \cap \Pi_{city}(PropertyForRent)$$

city
Aberdeen
London
Glasgow

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

Defines a relation consisting of the set of all tuples that are in both R and S.

Example - Cartesian Product

- List the names and comments of all clients who have viewed a property for rent.

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \times (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR56	PA14	too small
CR76	John	Kay	CR76	PG4	too remote
CR76	John	Kay	CR56	PG4	Blank
CR76	John	Kay	CR62	PA14	no dining room
CR76	John	Kay	CR56	PG36	Blank
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PG4	Blank
CR56	Aline	Stewart	CR62	PA14	no dining room
CR56	Aline	Stewart	CR56	PG36	Blank

Defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.

Example - Cartesian Product and Selection

- Use selection operation to extract those tuples where Client.clientNo = Viewing.clientNo.

$$\sigma_{\text{Client.clientNo}} = \text{Viewing.clientNo}((\prod_{\text{clientNo, fName, lName}}(\text{Client})) \times (\prod_{\text{clientNo, propertyNo, comment}}(\text{Viewing})))$$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	Comment
CR76	John	Kay	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	no dining room

- Cartesian product and Selection can be reduced to a single operation called a **Join.**

Join Operations (1 of 2)

- Join is a derivative of Cartesian product.
- Equivalent to performing a cross product followed by selection and projection.
- Most useful operations in RA
- One of the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMSs have intrinsic performance problems.

Join Operations (2 of 2)

- Various forms of join operation
  - Theta join
  - Equijoin (a particular type of Theta join)
  - Natural join
  - Outer join
  - Semijoin

Example - Natural Join

- List the names and comments of all clients who have viewed a property for rent.

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \bowtie (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

clientNo	fName	lName	propertyNo	comment
CR76	John	Kay	PG4	too remote
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA14	no dining room

An Equijoin of the two relations R and S over all common attributes *x*. One occurrence of each common attribute is eliminated from the result.

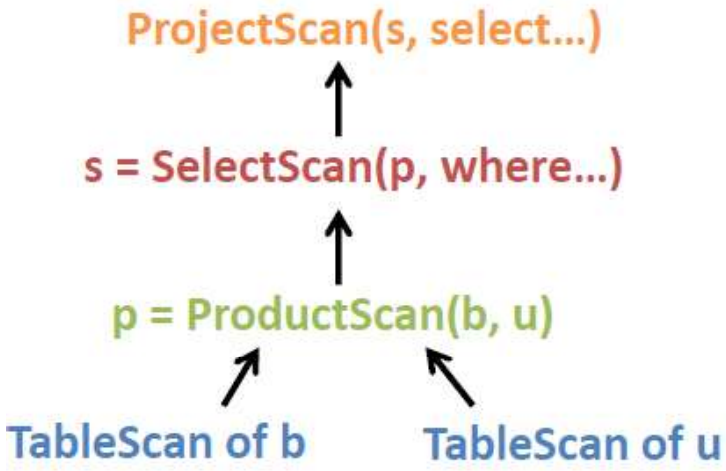
Planning & Scanning

Plan Before Scan

- A plan (tree) is a blueprint for evaluating a query
- Estimates cost by accessing statistics metadata only
  - No actual I/Os
  - Memory access only, ***very efficient***
- Once a good plan is decided, we then create a scan following the blueprint

Scan

- A ***scan*** represents the output of an operator in a relational algebra tree
  - I.e., output of a subtree (**partial query**)



Example 23.1 - Different Strategies (1 of 2)

Find all Managers who work at a London branch.

```
SELECT *
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
(s.position = 'Manager' AND b.city = 'London');
```

- Three equivalent RA queries are:

(1) $\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)} (Staff \times Branch)$

(2) $\sigma_{(position='Manager') \wedge (city='London')} (Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch)$

(3) $\sigma_{position='Manager'}(Staff) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'}(Branch))$

Example 23.1 - Different Strategies (3 of 3)

- Assume:
  - 1000 tuples in Staff; 50 tuples in Branch;
  - 50 Managers; 5 London branches;
  - no indexes or sort keys;
  - results of any intermediate operations stored on disk;
  - cost of the final write is ignored;
  - tuples are accessed one at a time.

Example 23.1 - Cost Comparison

- Cost (in disk accesses) are:

(1) $(1000+50)+2*(1000*50)=101\ 050$ $\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)}$ (Staff X Branch)

(2) $2*1000+(1000+50)=3\ 050$ $\sigma_{(position='Manager') \wedge (city='London')}$(Staff $\bowtie_{Staff.branchNo=Branch.branchNo}$ Branch)

(3) $1000+2*50+5+(50+5)=1\ 160$ $\sigma_{position='Manager'}$(Staff)) $\bowtie_{Staff.branchNo=Branch.branchNo}$ ($\sigma_{city='London'}$ (Branch))

- Cartesian product and join operations much more expensive than selection, and third option significantly reduces size of relations being joined together.

Tutorial 5: explaining and analyzing query execution plans in PostgreSQL

Why does it matter?

- A good scan tree can be faster than a bad one for orders of magnitudes

Which cost to estimate?

- CPU delay, memory delay, or I/O delay?
- The ***number of block accesses*** performed by a scan is usually the most important factor in determining running time of a query

How to find a Good Plan Tree?

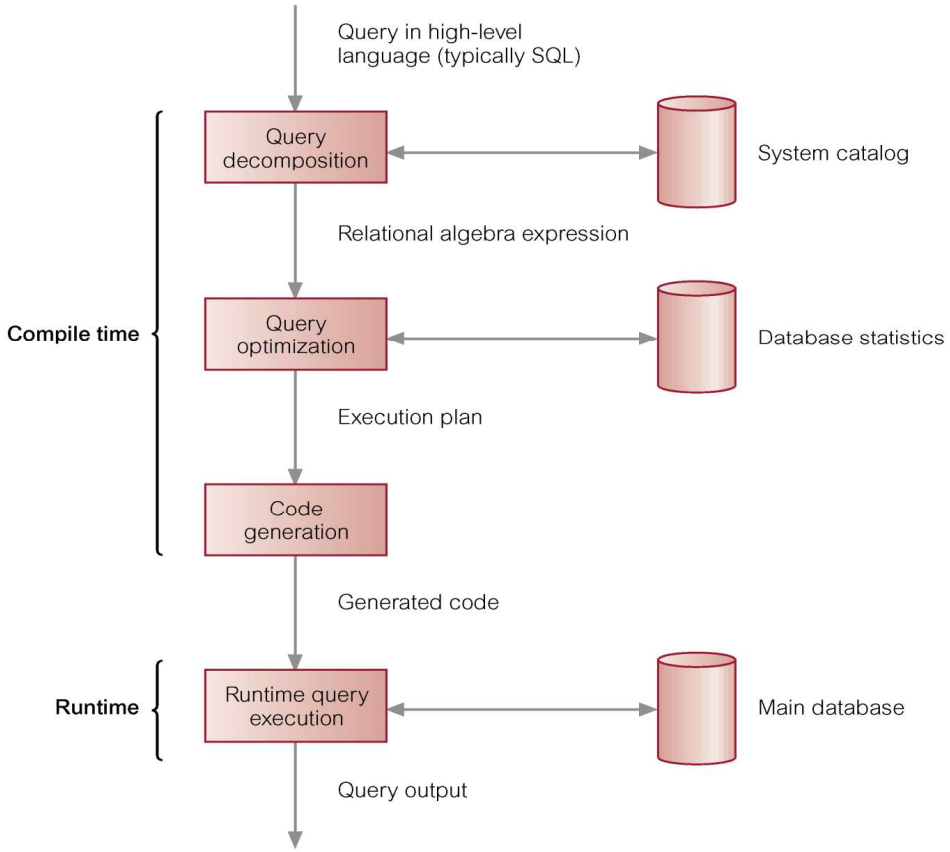
- The planner can create multiple trees first, and then pick the one having the lowest cost
- Determining the best plan tree for a SQL command is called ***planning***

Query Processing

Query Processing

- Activities involved in retrieving data from the database.
- Aims of QP:
  - transform query written in high-level language (e.g. SQL), into correct and efficient execution strategy expressed in low-level language (implementing RA);
  - execute strategy to retrieve required data.

Phases of Query Processing



Query Decomposition

- Aims are to transform high-level query into RA query and check that query is syntactically and semantically correct.
- Typical stages are:
  - Analysis (***Parsing)***,
  - normalization,
  - semantic analysis (***Verification)***,
  - simplification,
  - query restructuring.

Analysis - Example

```
SELECT staffNumber
FROM Staff
WHERE position > 10;
```

- This query would be rejected on two grounds:
  - staffNumber is not defined for Staff relation (should be staffNo).
  - Comparison '>10' is incompatible with type position, which is variable character string.

Normalization

- Converts query into a normalized form for easier manipulation.
- Predicate can be converted into one of two forms:

Conjunctive normal form:

$$(position = \text{'Manager'} \vee salary > 20000) \wedge (branchNo = \text{'B003'})$$

Disjunctive normal form:

$$(position = \text{'Manager'} \wedge branchNo = \text{'B003'}) \vee$$
$$(salary > 20000 \wedge branchNo = \text{'B003'})$$

Semantic Analysis (1 of 2)

- Rejects normalized queries that are incorrectly formulated or contradictory.
- Query is **incorrectly formulated** if components do not contribute to generation of result.
- Query is **contradictory** if its predicate cannot be satisfied by any tuple.
- Algorithms to determine correctness exist only for queries that do not contain disjunction and negation.

Semantic Analysis (2 of 2)

- For these queries, could construct:
  - A relation connection graph.
  - Normalized attribute connection graph.

Relation connection graph

Create node for each relation and node for result. Create edges between two nodes that represent a join, and edges between nodes that represent projection.

- If not connected, query is incorrectly formulated.

Simplification

- Detects redundant qualifications,
- Eliminates common sub-expressions,
- Transforms query to semantically equivalent but more easily and efficiently computed form.
  - Typically, access restrictions, view definitions, and integrity constraints are considered.
  - Assuming user has appropriate access privileges, first apply well-known idempotency rules of boolean algebra.

Syntax vs Semantic

- The ***syntax*** of a language is a set of rules that describes the strings that could possibly be meaningful statements
- Is this statement syntactically legal?

```
SELECT FROM TABLES t1 AND t2 WHERE b - 3;
```

- No
  - SELECT clause must refer to some field
  - TABLES is not a keyword
  - AND should be separated predicates not tables
  - B-3 is not a predicate

Syntax vs Semantic

- Is this statement syntactically legal?

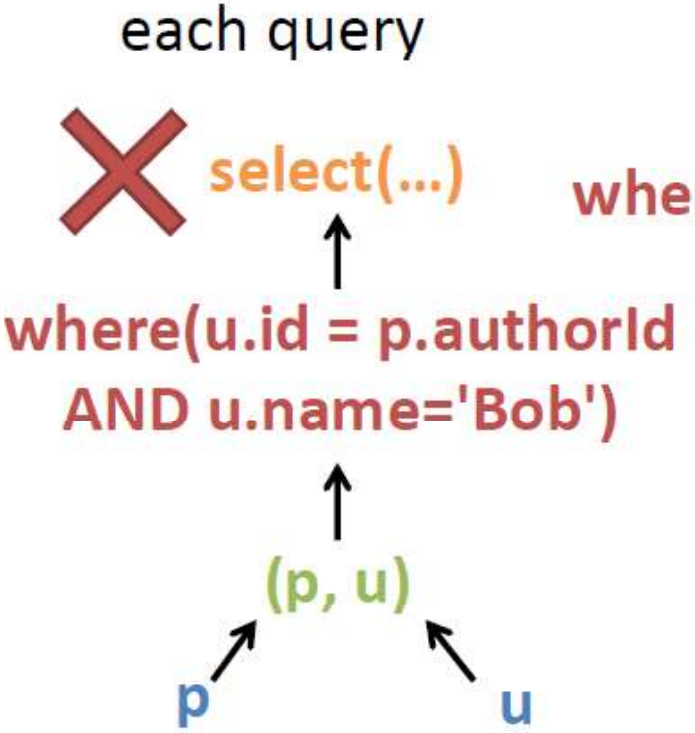
```
SELECT a FROM t1, t2 WHERE b = 3;
```

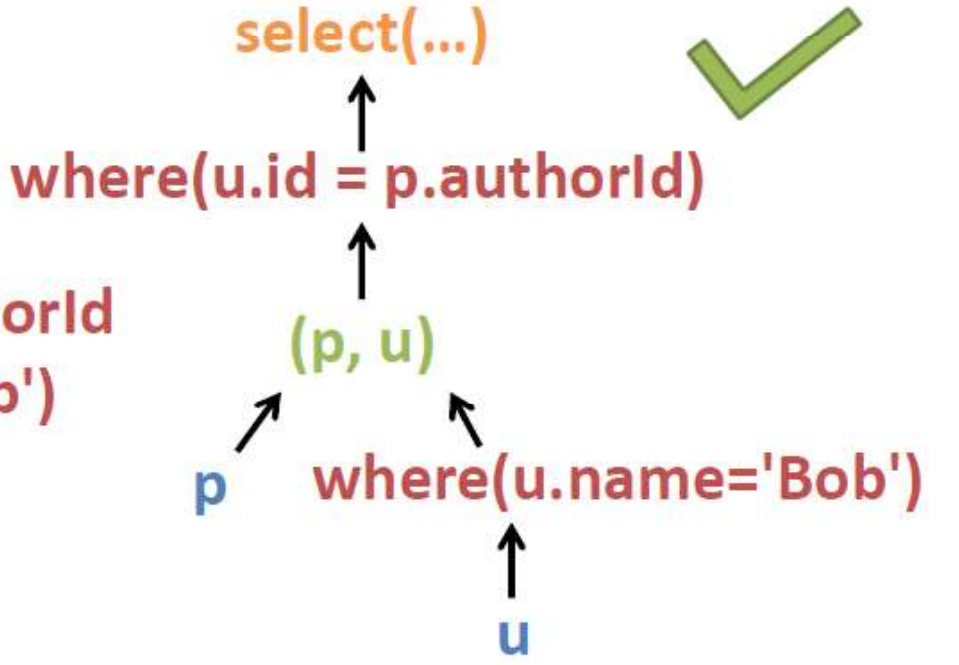
- Yes, we can infer that this statement is a query
- But is it actually meaningful?
- The ***semantics*** of a languages specifies the actual meaning of a syntactically correct string
- Whether it is semantically legal depends on
  - Is `a` a field name?
  - Are `t1`, `t2` the names of tables?
  - Is `b` the name of a numeric field?
  - Semantic information is stored in the database's metadata (catalog/dictionary)

Query Optimization

Query Optimization

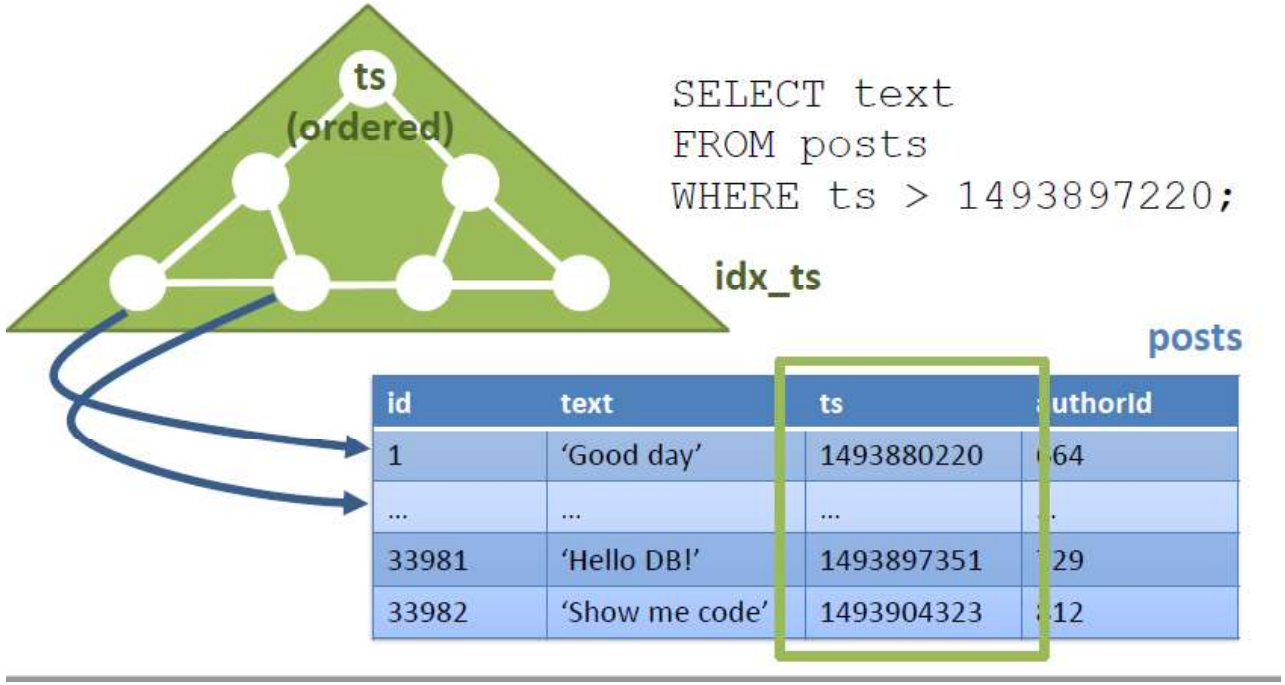
- ***Planning***: DBMS finds the best ***plan tree*** for each query





Query Optimization

- ***Indexing*** creates a search tree for columns (s)



What is Query Optimization?

- Query optimization is the activity of choosing an efficient query execution plan.
  - Systematic estimation of the cost of different query execution plans and choosing the one with the lowest cost
  - Estimation of plan cost is based on statistical information about relations (e.g., number of tuples, number of distinct values, etc.)
- It is about making decisions about data access methods
- It depends on the file organization

How it works?

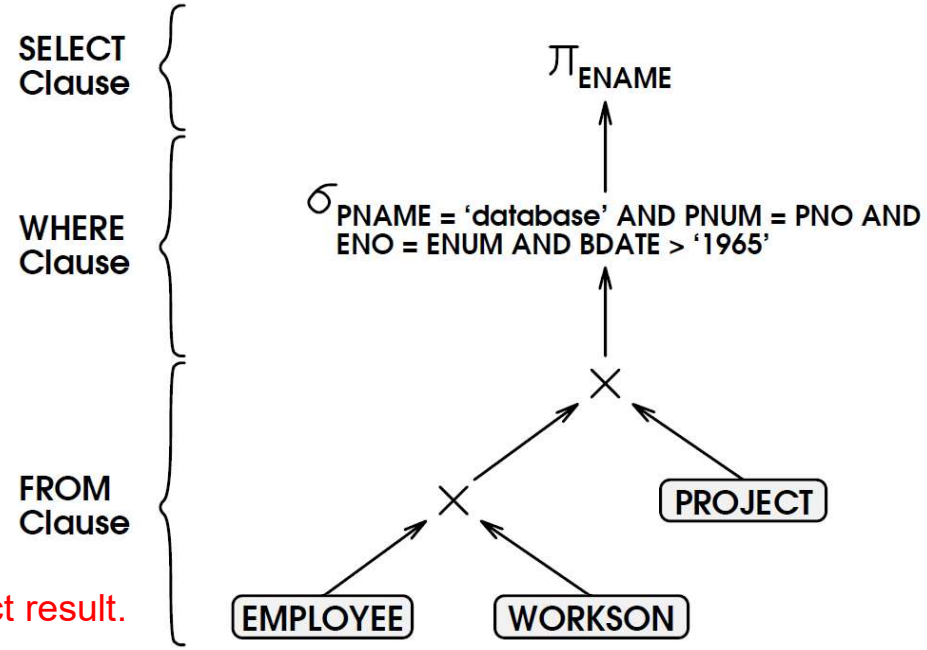
- SQL queries are decomposed into query blocks
- Each query block is translated to an equivalent relational algebra expression and represented as a query tree of logical operators
- Blocks are separately optimized (cost-based query optimization)
  - Cost estimates should consider disk accesses and cardinality of the relation (tuples)

Query Optimization Example 1

- SQL query transform it into corresponding relational algebra query trees

SELECT ENAME
FROM EMPLOYEE, WORKSON, PROJECT
WHERE PNAME = '*database*' AND
PNUM = PNO AND
ENO = ENUM AND
BDATE > '1965'

Canonical tree



This query tree (procedure) will compute the correct result.
However, the performance will be very poor.
→ *needs optimization !*

Optimization Strategies

GOAL: Reducing the sizes of the intermediate results as quickly as possible.

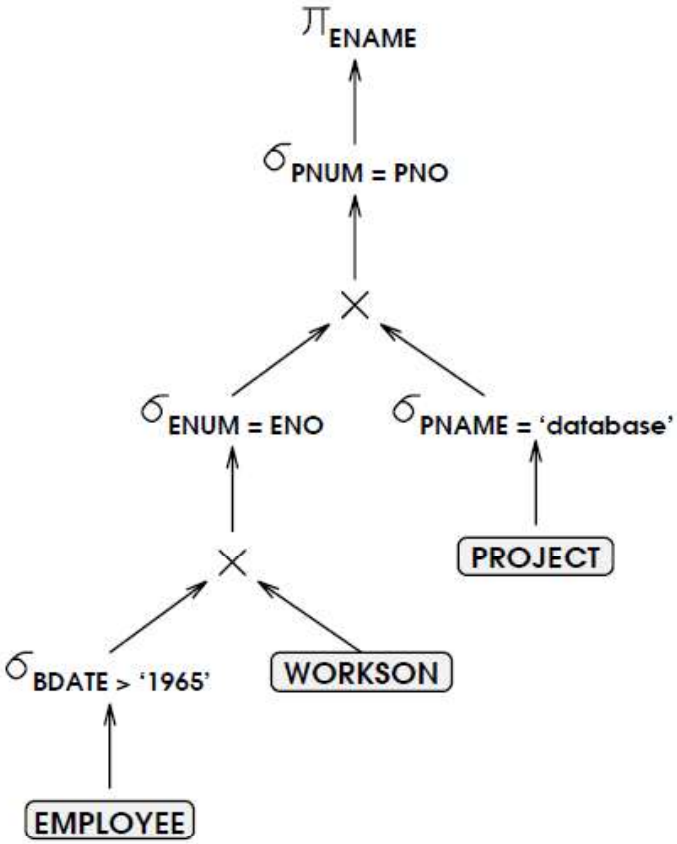
STRATEGY:

1. Move SELECTs and PROJECTs as far down the query tree as possible.
2. Among SELECTs, reordering the tree to perform the one with lowest selectivity factor first.
3. Among JOINs, reordering the tree to perform the one with lowest join selectivity first.

Optimized Tree 1

Apply SELECTs First

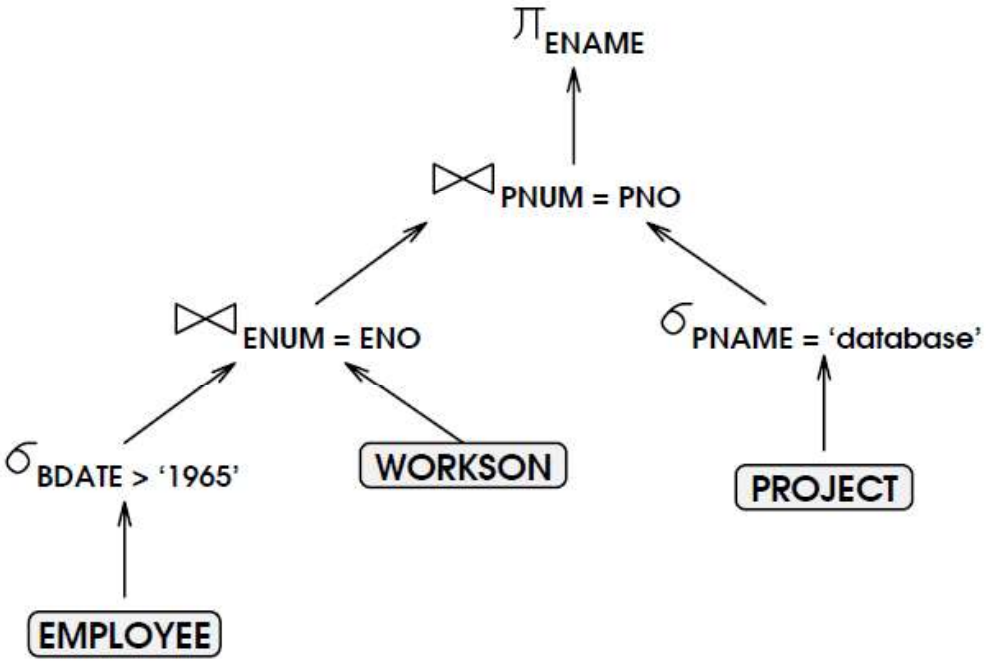
SELECT ENAME
FROM EMPLOYEE, WORKSON, PROJECT
WHERE PNAME = '*database*' AND
PNUM = PNO AND
ENO = ENUM AND
BDATE > '1965'



Optimized Tree 2

Replace "σ −×" by " ⋈ "

SELECT ENAME
FROM EMPLOYEE, WORKSON, PROJECT
WHERE PNAME = '*database*' AND
PNUM = PNO AND
ENO = ENUM AND
BDATE > '1965'



Optimized Tree 3

Move PROJECTs Down

SELECT ENAME
FROM EMPLOYEE, WORKSON, PROJECT
WHERE PNAME = '*database*' AND
PNUM = PNO AND
ENO = ENUM AND
BDATE > '1965'