D2.P2: A Query Planner Based on Quantitative and Structural Information

Francesco Scarcello
Alfredo Mazzitelli
Università della Calabria

Query Planners

- Given a query and some information on the database, find the best query plan
- A hard problem, if the query involves many relations
- Populating and refreshing data warehouses often requires a number of quite long batch queries on the reconciled database

- Many techniques
  - Quantitative methods
  - Structural methods

- Our prototype combine these separate worlds

The Prototype

- Takes as its input a query (SQL or Datalog) and quantitative information on the data
- Outputs:
  - An optimal hypertree decomposition HD
  - A script representing a query plan corresponding to HD (with some hints for the Oracle optimizer)

- Note that it deals with non-Boolean queries, possibly containing aggregate operators

Two different approaches

1. Data oriented:
   – Maintain a dictionary with information on the data
   – Exploit this information for finding the best query plan
2. Structure Oriented:
   – Look for structural properties of the query
   – Possibly, answer the query in (output) polynomial time

What is actually used?

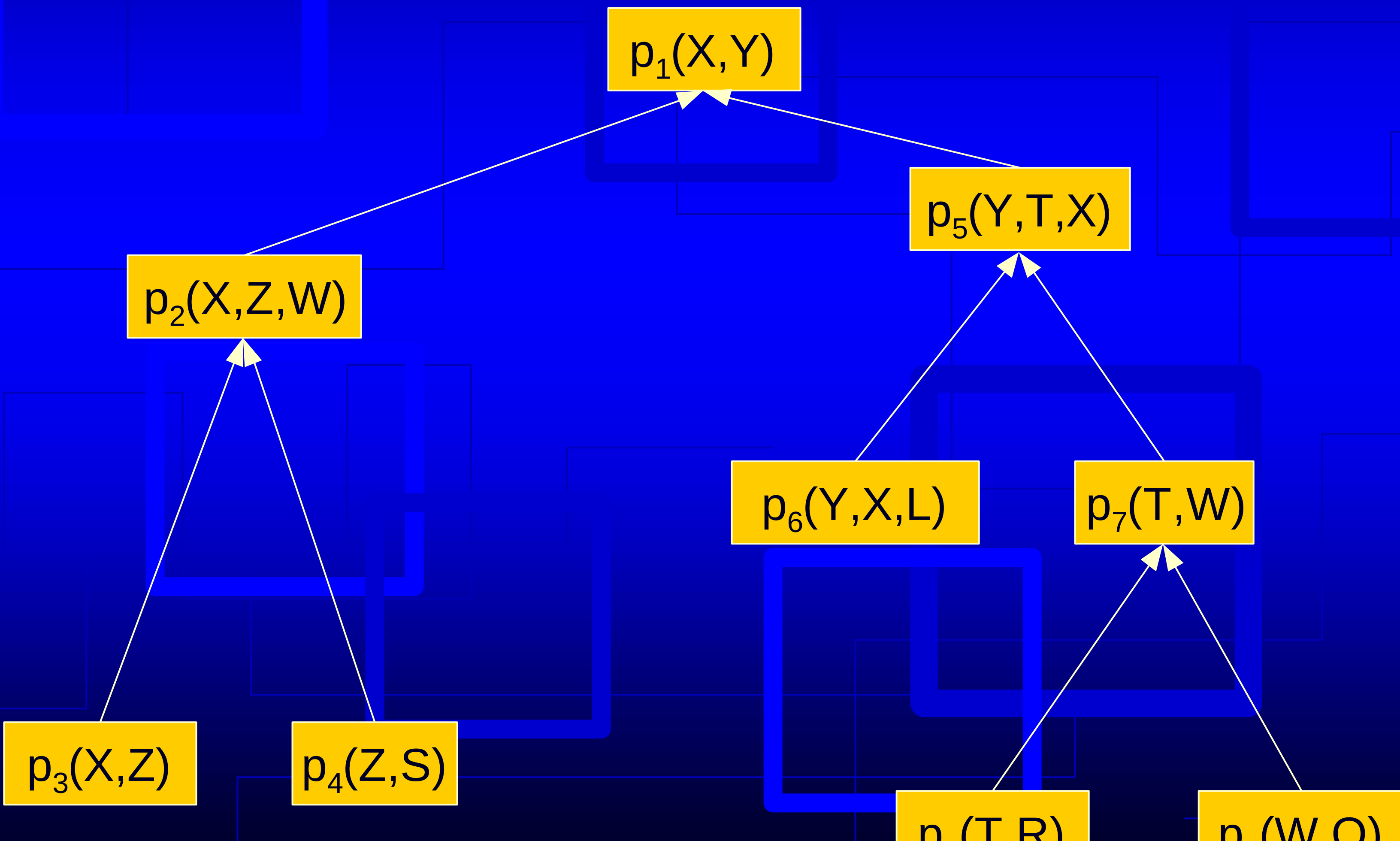
- All commercial DBMS are equipped with Data-oriented query planners
- Many important theoretical papers deals with structural properties of queries

⇒ Wide tractability classes have been identified

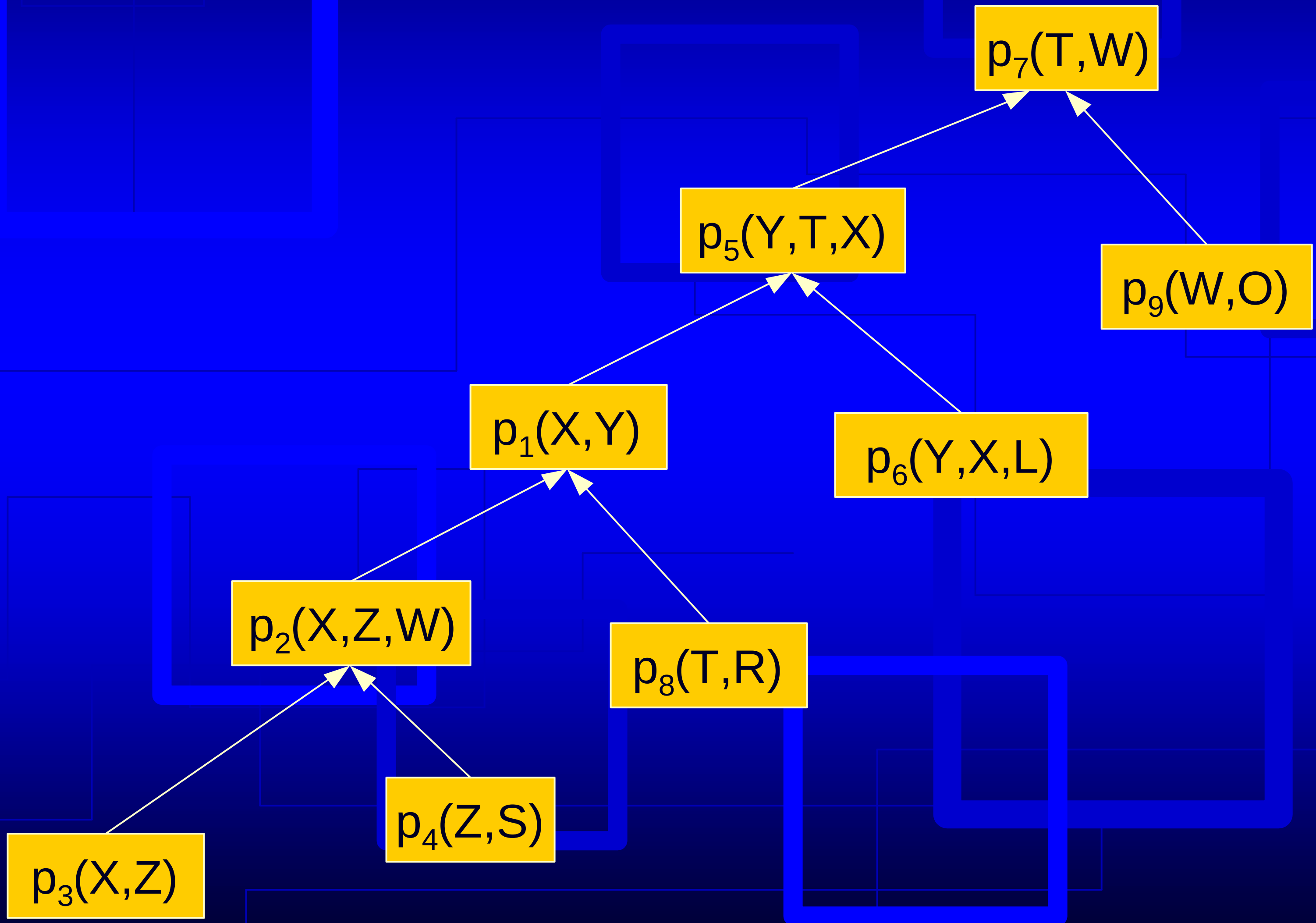
Why?

- Data Oriented:
  - Good amount of statistical information available
  - Efficient algorithms (mainly heuristics)
  - Simple kind of queries (typically short queries)
- Structure Oriented:
  - Polynomial Upper Bound on the cost of answering the query
  - Very good for long queries
  - Able to deal with databases without information on the data

What is the best choice?



What is the best choice?



Some data oriented planners

- Dynamic programming
- Greedy
- Iterative Dynamic Programming

  [Kossmann & Stoker, TODS, 2000]
- Restrictions of the search space:

		Left-deep tree		*Bushy tree*
9 atomi	→	$n!$	→	$\binom{2(n-1)}{n-1} \cdot (n-1)!$
		362.880		518.918.400

Structural methods: acyclicity

- The answer of an acyclic (conjunctive) query can be computed in output polynomial time
  [Yannakakis, VLDB'81]

- An Acyclic Boolean Conjuctive Query (***ABCQ***) Q may be answered in $O(|T|\, n \log n)$, where *T* is a *join tree* for *Q* and *n* is the size of the largest relation

- ABCQ is LOGCFL-complete and thus is highly parallelizable [GLS 98]

- Yannakakis's Algorithm is based on the use of **semi-join**

How to deal with such a long query?

$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$

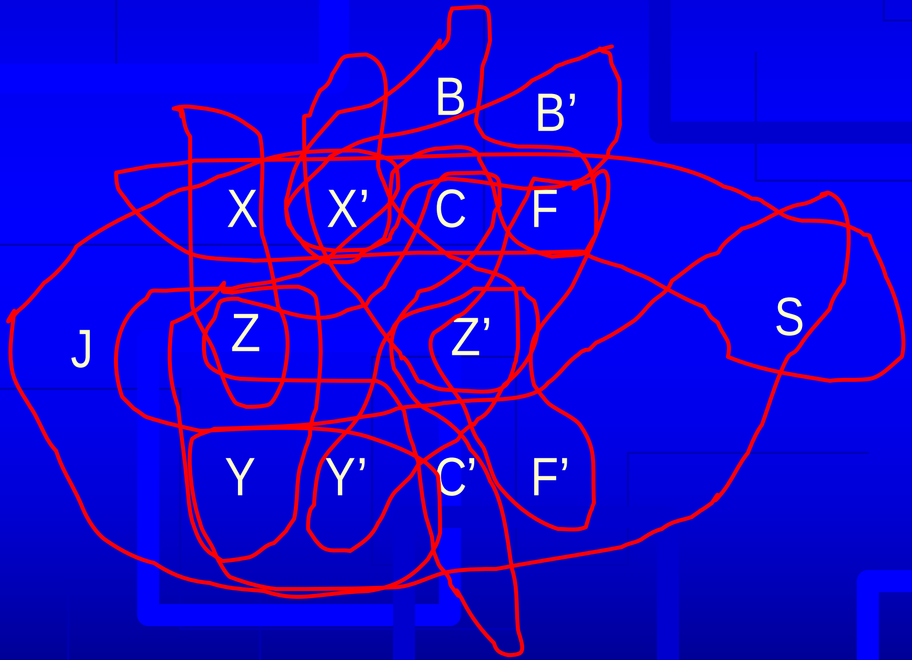
m = 11 !

n	size of the database
m	number of atoms in the query

- Classical methods worst-case complexity: $O(n^m)$

Look at the hypergraph…

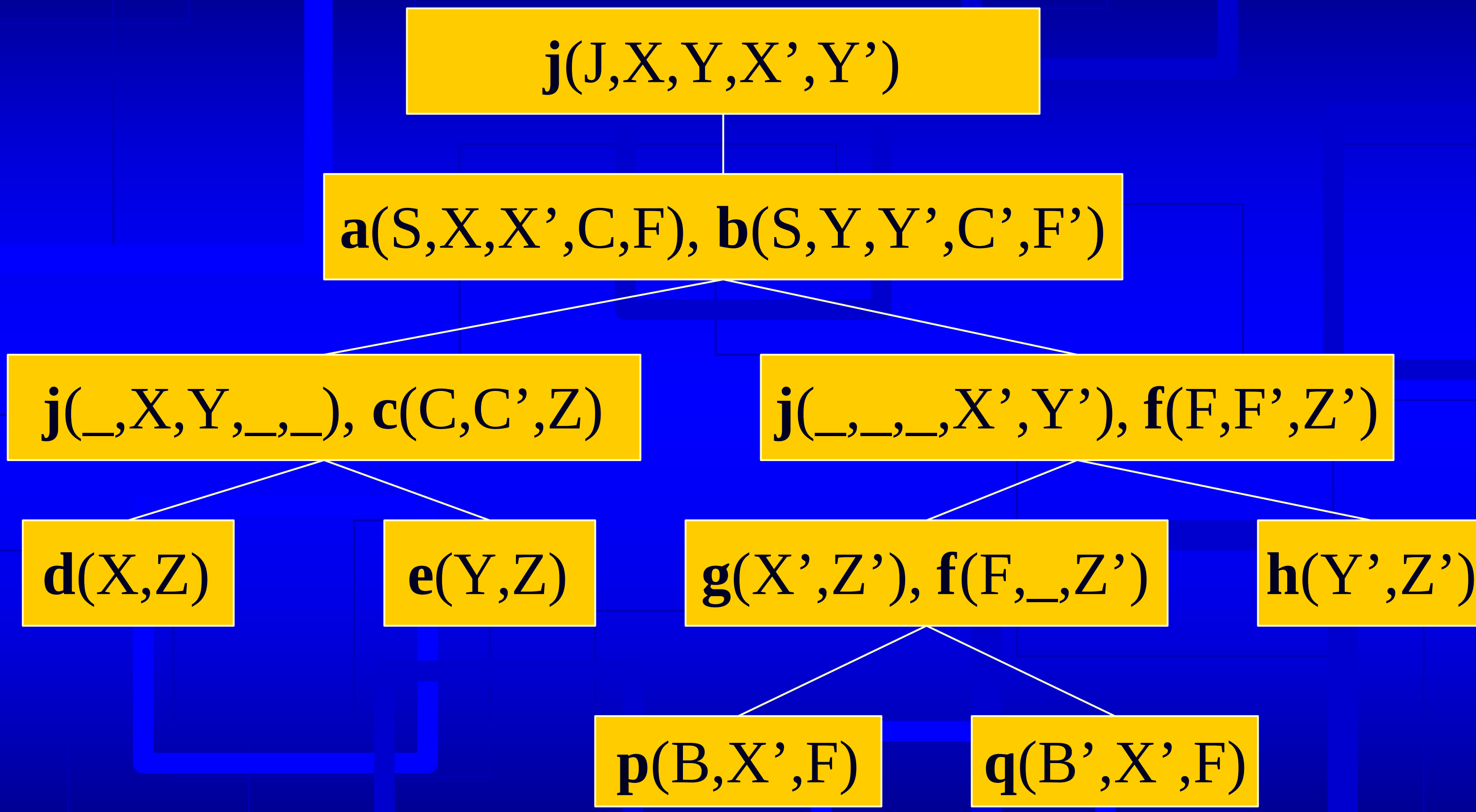
$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge$$
$$e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge$$
$$j(J,X,Y,X',Y') \wedge p(B,X',F) \wedge q(B',X',F)$$



Despite its apparence, this query is nearly acyclic

It can be evaluated in $O(m \cdot n^{2} \cdot \log n)$

Hypertrees and Query Plans



It is just a partial specification!

A quantitative extension for HYPERTREE

- A new method, which exploits quantitative and structural information
- A cost model for evaluating plans determined by hypertree decompositions
- An algorithm for computing optimal bounded hypertree decompositions:
  find the NF hypertree decomposition of width at most *k* such that its associated cost is the minimum over all *k*-bounded NF hypertree decompositions

A cost model for Hypertree

The cost of a **subtree** rooted at ***p***

$$cost(T_p) \triangleq cost(p)$$
$$+ \sum_{c \in children(p)} cost(T_c)$$
$$+ \sum_{c \in children(p): c(c) - c(p) \neq \varnothing} \left| \pi_{c(p)}(S_c) \right|$$
$$+ cost(E(S_p))$$

$$S_p \triangleq \begin{cases} J_p, \forall p \in leaves(T_p) \\ J_p \bowtie_{c \in children(p)} \pi_{c(p)}(S_c), & \text{otherwise} \end{cases}$$

Example

Consider again the conjunctive query:

$$ans \leftarrow a(S,X,Xp,C,F) \wedge b(S,Y,Yp,Cp,Fp) \wedge c(C,Cp,Z) \wedge \wedge d(X,Z) \wedge e(Y,Z) \wedge f(F,Fp,Zp) \wedge \wedge g(Xp,Zp) \wedge h(Yp,Zp) \wedge j(J,X,Y,Xp,Yp).$$

Quantitative Information

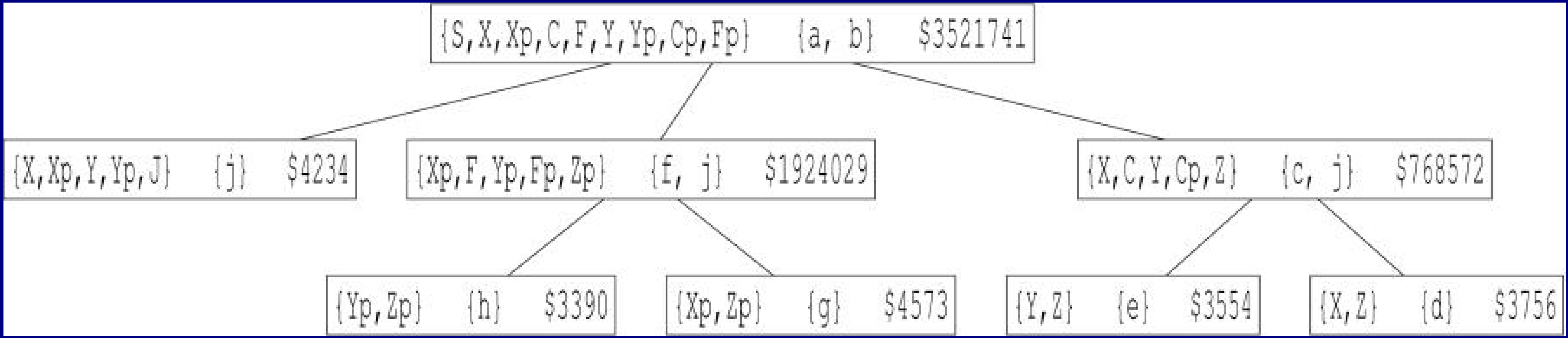
<ATOM>	<CARDINALITY>
a(S,X,Xp,C,F)	4606
b(S,Y,Yp,Cp,Fp)	2808
c(C,Cp,Z)	1748
d(X,Z)	3756
e(Y,Z)	3554
f(F,Fp,Zp)	2892
g(Xp,Zp)	4573
h(Yp,Zp)	3390
j(J,X,Y,Xp,Yp)	4234

Cardinality of relations in the Database

<ATOM>	<VARIABLE>	<SELECTIVITY>
a(S,X,Xp,C,F)	S	14
a(S,X,Xp,C,F)	X	24
a(S,X,Xp,C,F)	Xp	16
a(S,X,Xp,C,F)	C	21
a(S,X,Xp,C,F)	F	15
b(S,Y,Yp,Cp,Fp)	S	17
b(S,Y,Yp,Cp,Fp)	Y	5
b(S,Y,Yp,Cp,Fp)	Yp	12
b(S,Y,Yp,Cp,Fp)	Cp	20
b(S,Y,Yp,Cp,Fp)	Fp	7
c(C,Cp,Z)	C	18
c(C,Cp,Z)	Cp	7
c(C,Cp,Z)	Z	19
d(X,Z)	X	18
d(X,Z)	Z	7
e(Y,Z)	Y	21
e(Y,Z)	Z	13

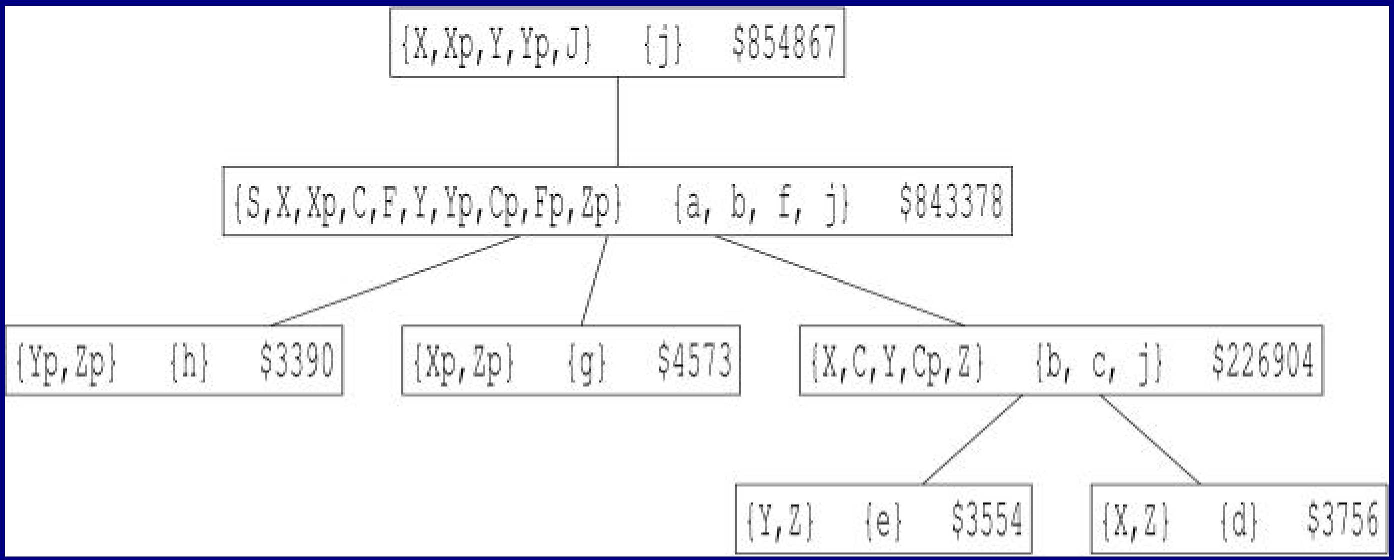
Attribute selectivity

Example: k = 2



k = 2 → 3521741 *width = 2*

Example: k = 4



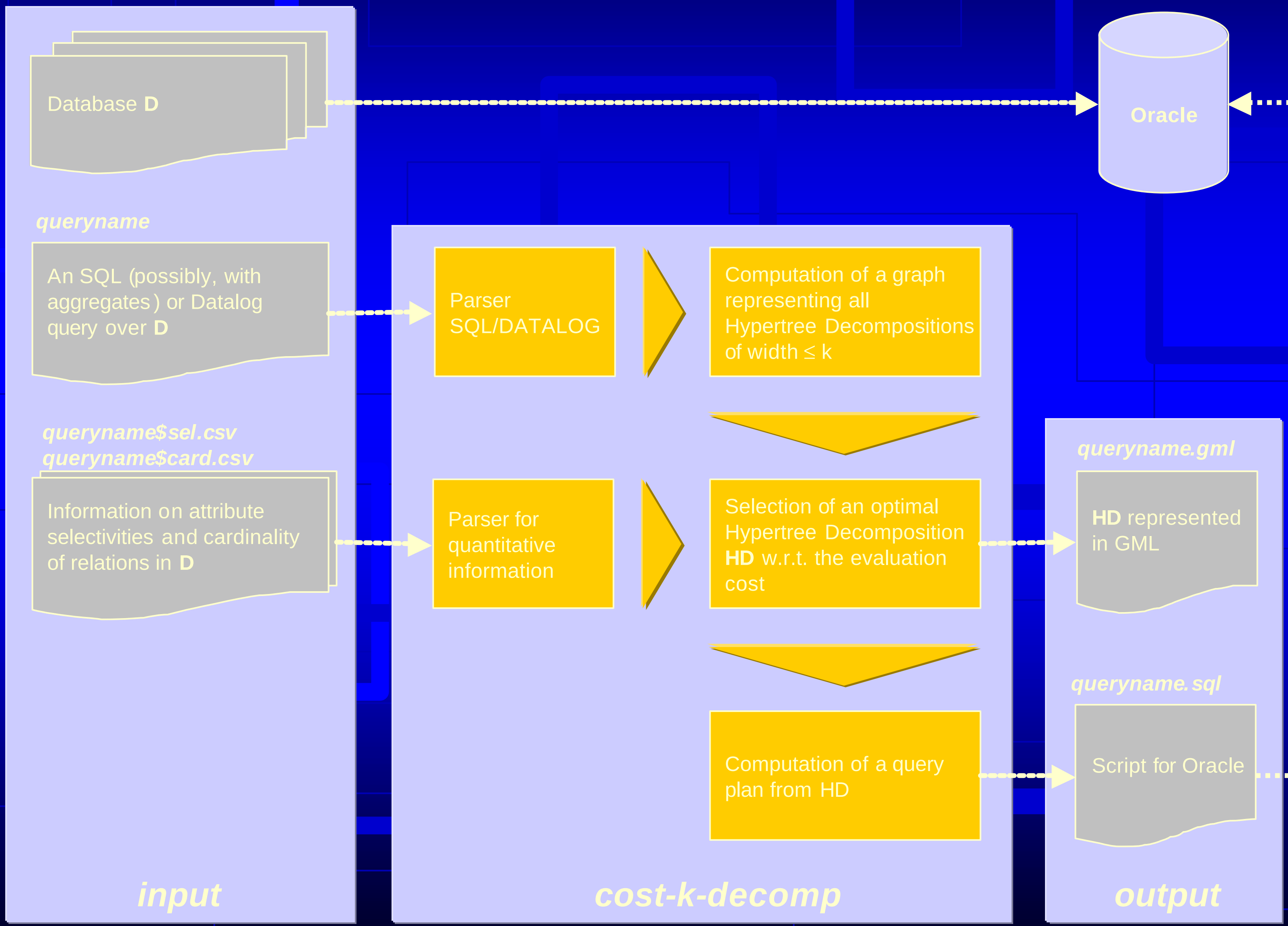
k = 4 → 854867 *width = 4*

In Summary

The prototype implements a new method for computing query plans based on hypertree decomposition:

- Exploits quantitative information on the data
- Exploits the structure of the query
- Guarantees a polynomial-time upper bound
- Quality/Efficiency Ratio may be tuned through the parameter *k*
- The algorithm is parallelizable
- It is not Fixed-Parameter Tractable

The Prototype Architecture

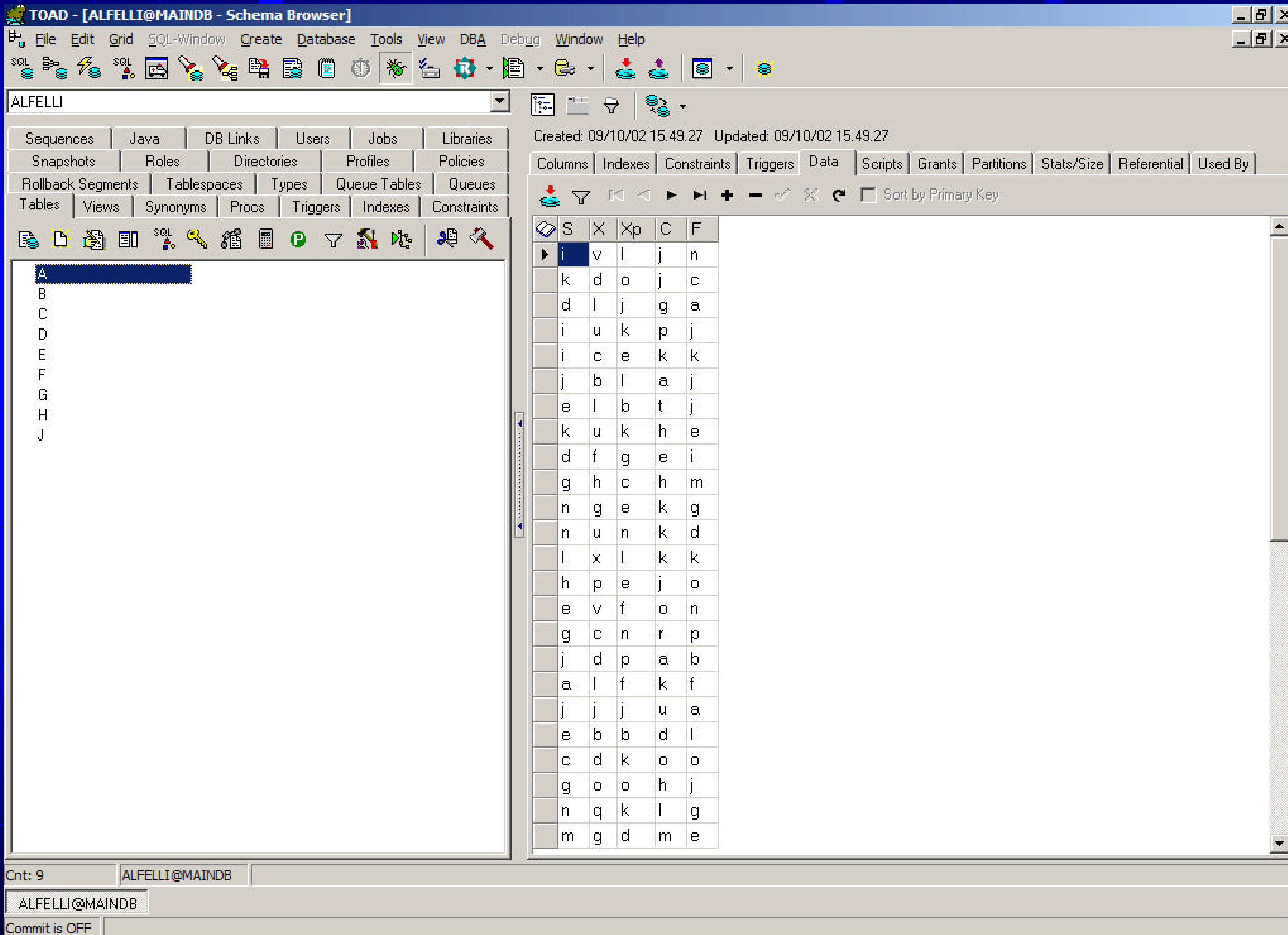


Example: Input Query

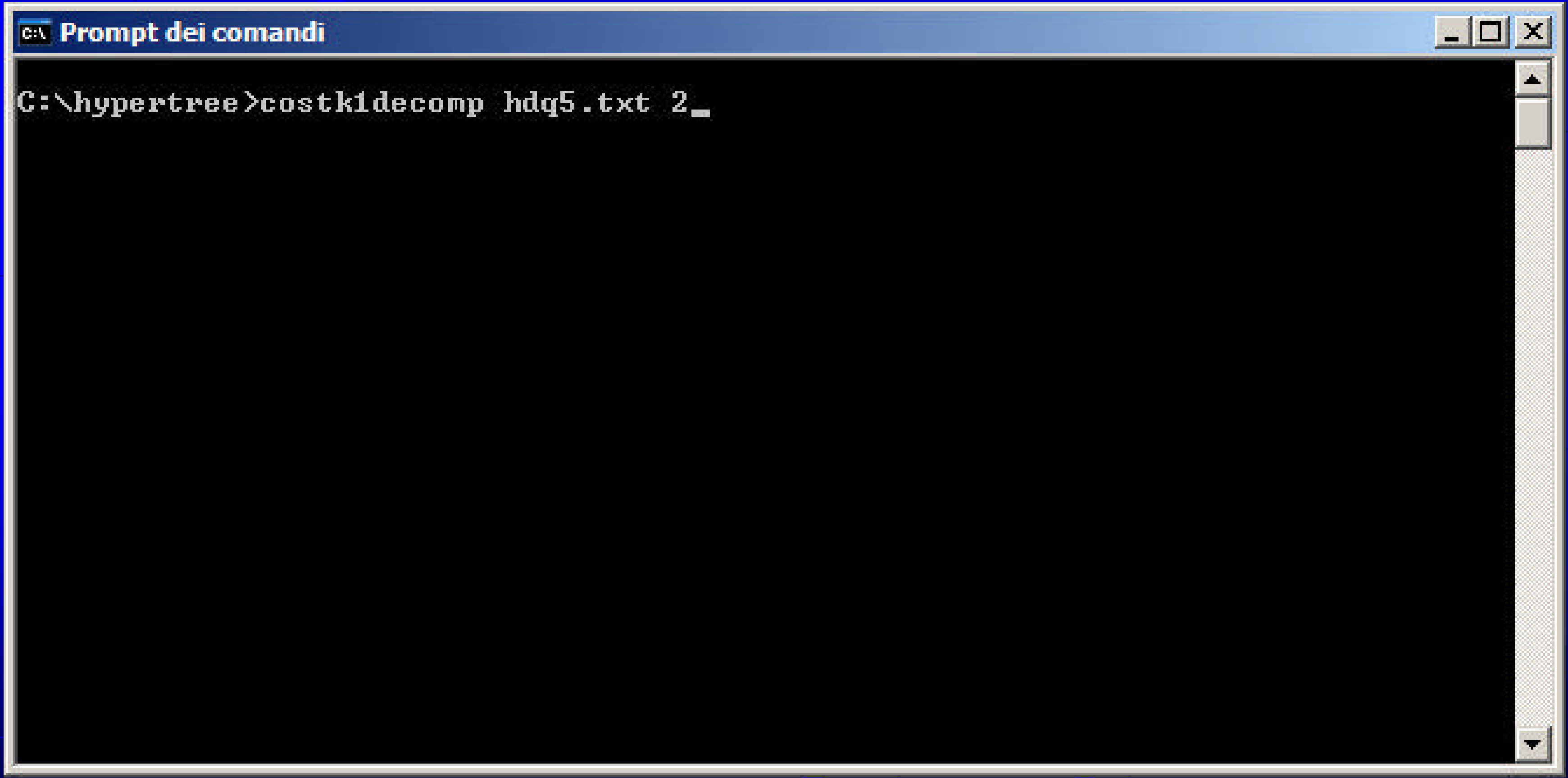
DATALOG	***SQL***
out(X,Y):- a(S,X,XP,C,F), b(S,Y,YP,CP,FP), c(C,CP,Z), d(X,Z), e(Y,Z), f(F,FP,ZP), g(XP,ZP), h(YP,ZP), j(J,X,Y,XP,YP).	SELECT a.X,b.Y FROM a,b,c,d,e,f,g,h,j WHERE a.S=b.S AND a.C=c.C AND …

hdq5.txt

Example: Input Database



Running the prototype

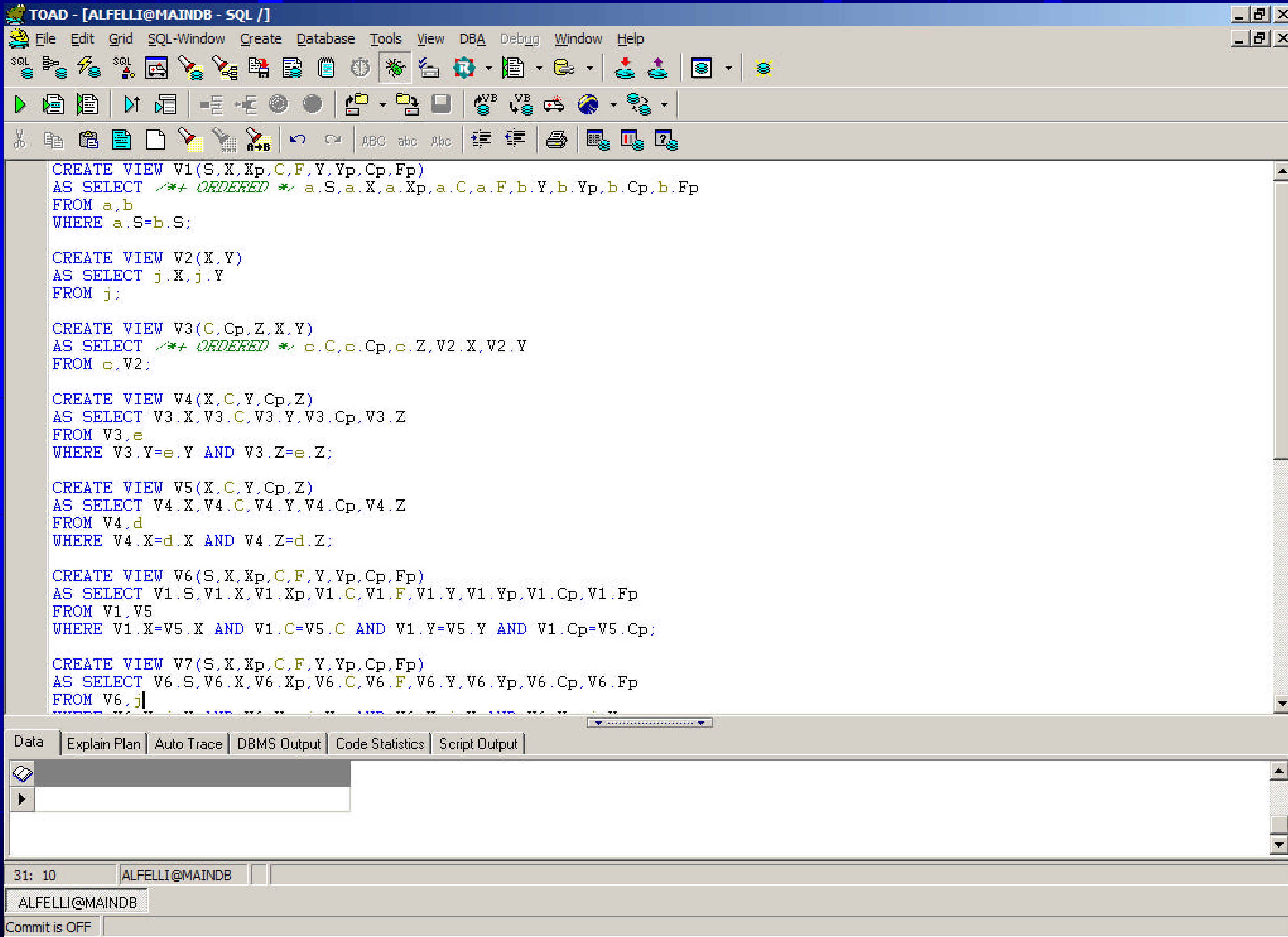


Some output information

Prompt dei comandi

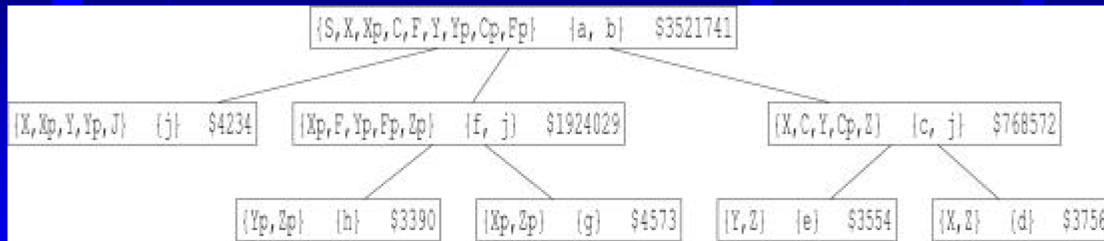
```
C:\hypertree>costk1decomp hdq5.txt 2

query file     = hdq5.txt
upper bound    = 2
representation = non verbose

running ... elapsed time (sec): 0.01

>> query atoms                      = 9
>> query variables                  = 12

>> OPTIMAL COST BASED HYPERTREE DECOMPOSITION FOUND!

>> cost                             = 3521741
>> width                            = 2
>> depth                            = 3
>> vertices                         = 8
>> OUTPUT                           = hdq5.txt.gml, hdq5.txt.sql

C:\hypertree>
```

A script for Oracle (with hints)



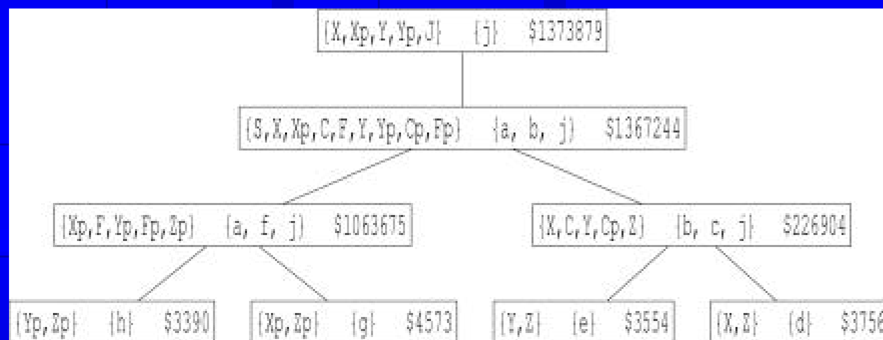
Some preliminary results

k = 2



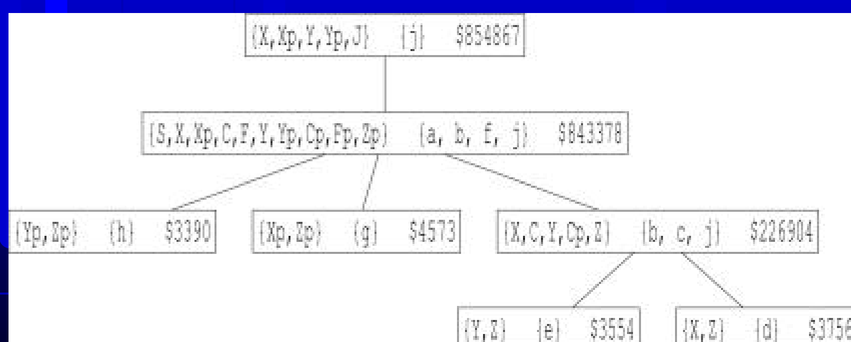
0.6

k = 3



0.04

k = 4 …



0.02