AI infusions in Db2

Date: May 12, 2026

Speakers

Raymond Yao: Senior Manager – Db2 AI & Query Compiler

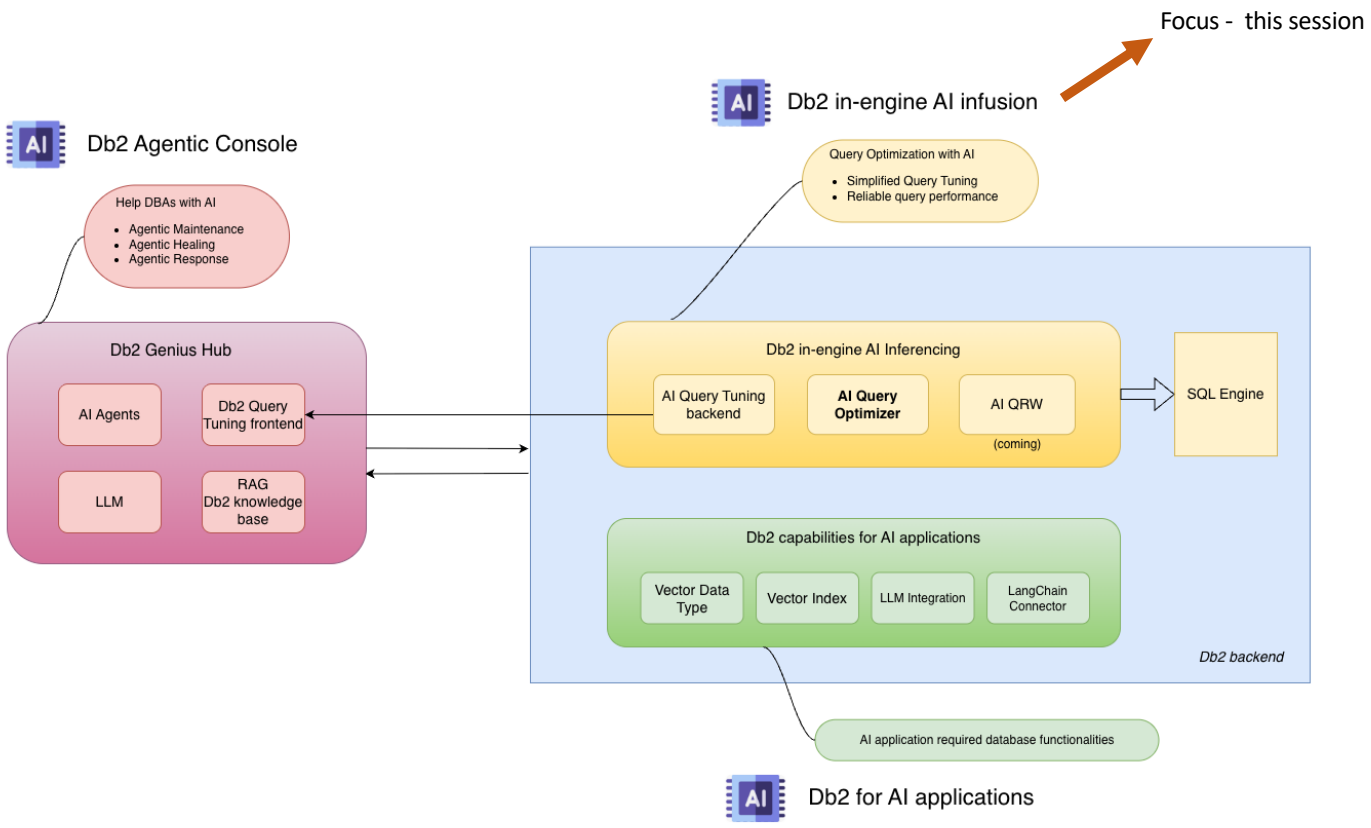
Vincent Corvinelli: Senior Dev, Team Lead – Db2 AI & Query Compiler

Lori Strain: Senior Developer – Db2 Query Optimizer

Agenda

- Db2 AI dimensions
- Db2 in-engine AI infusion Client Value
- AI Query Tuning
- AI Query Optimizer
- Future Work

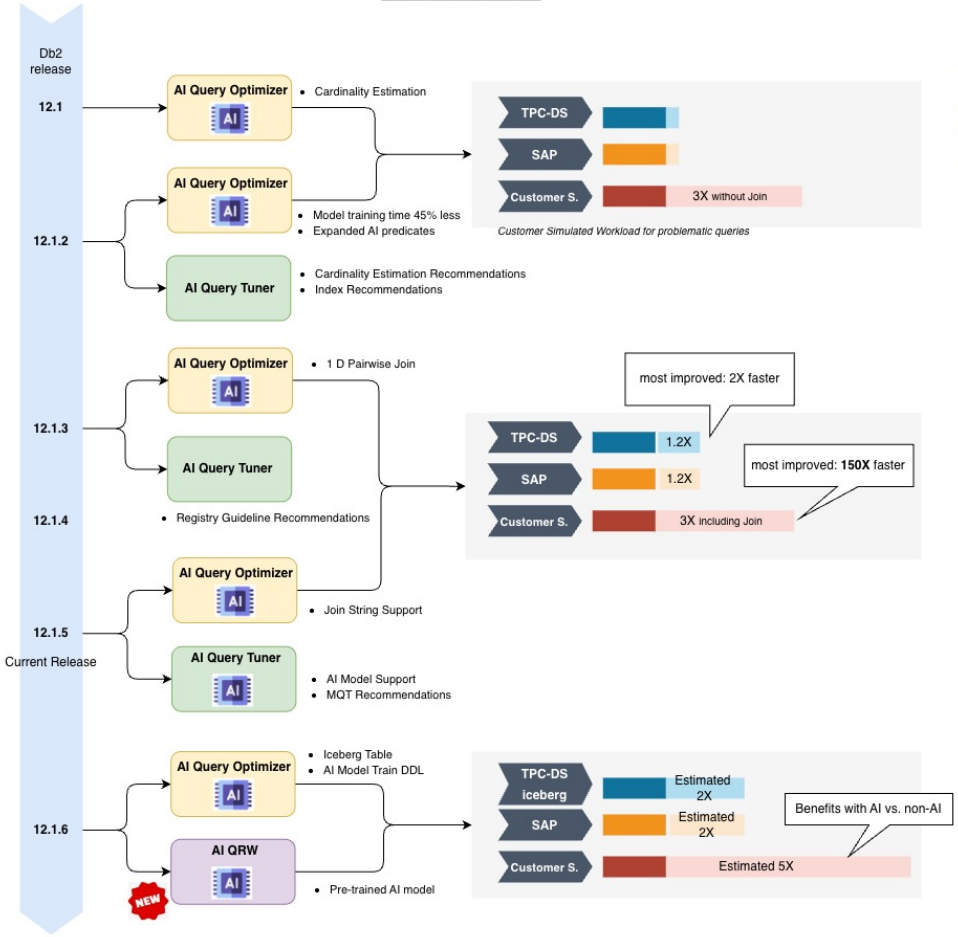
Db2 AI dimensions



Db2 in-engine AI infusion Customer Value

Making Query Optimization **Simple** While Achieving **Reliable** and **Stable** Query Execution **Performance**

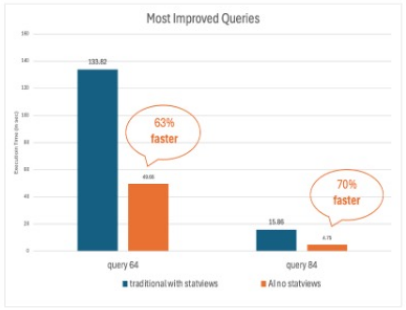
Performance



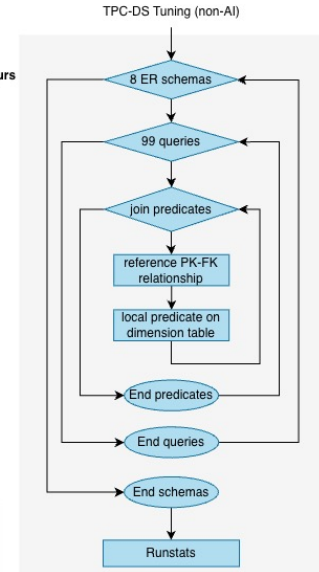
Simplicity

statviews is manually **tuned** with industry **expertise** and a lot of **hours**

AI infused that dramatically reduces the need for tuning



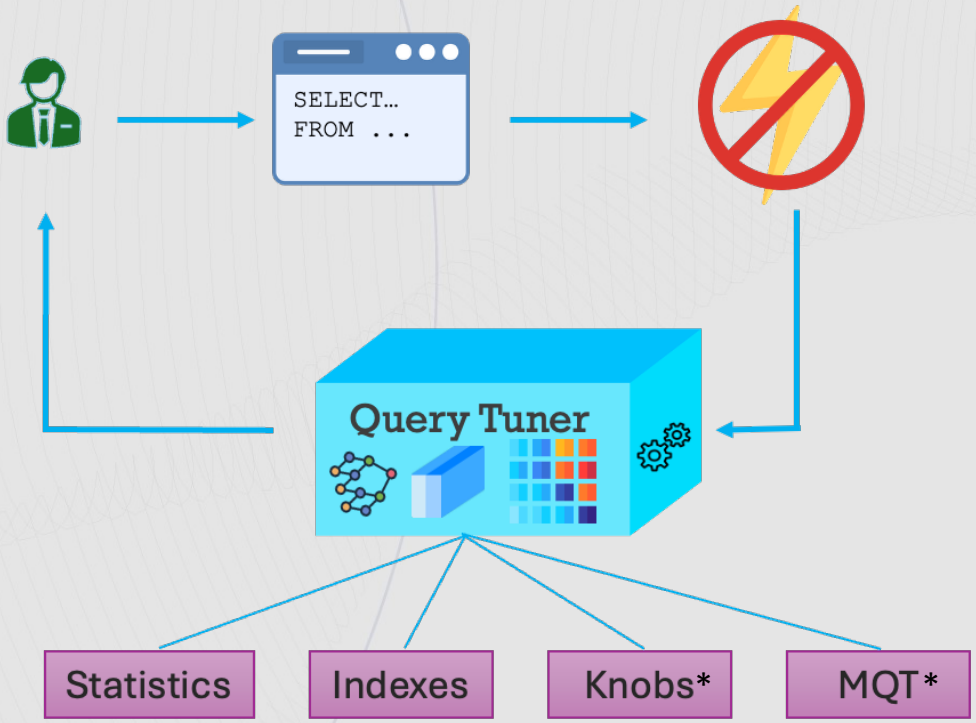
- Tuning time saved: 1000% from a senior DBA
- Customer cases: issues requiring statview won't appear



Non-AI: **3240** minutes (54 hours)

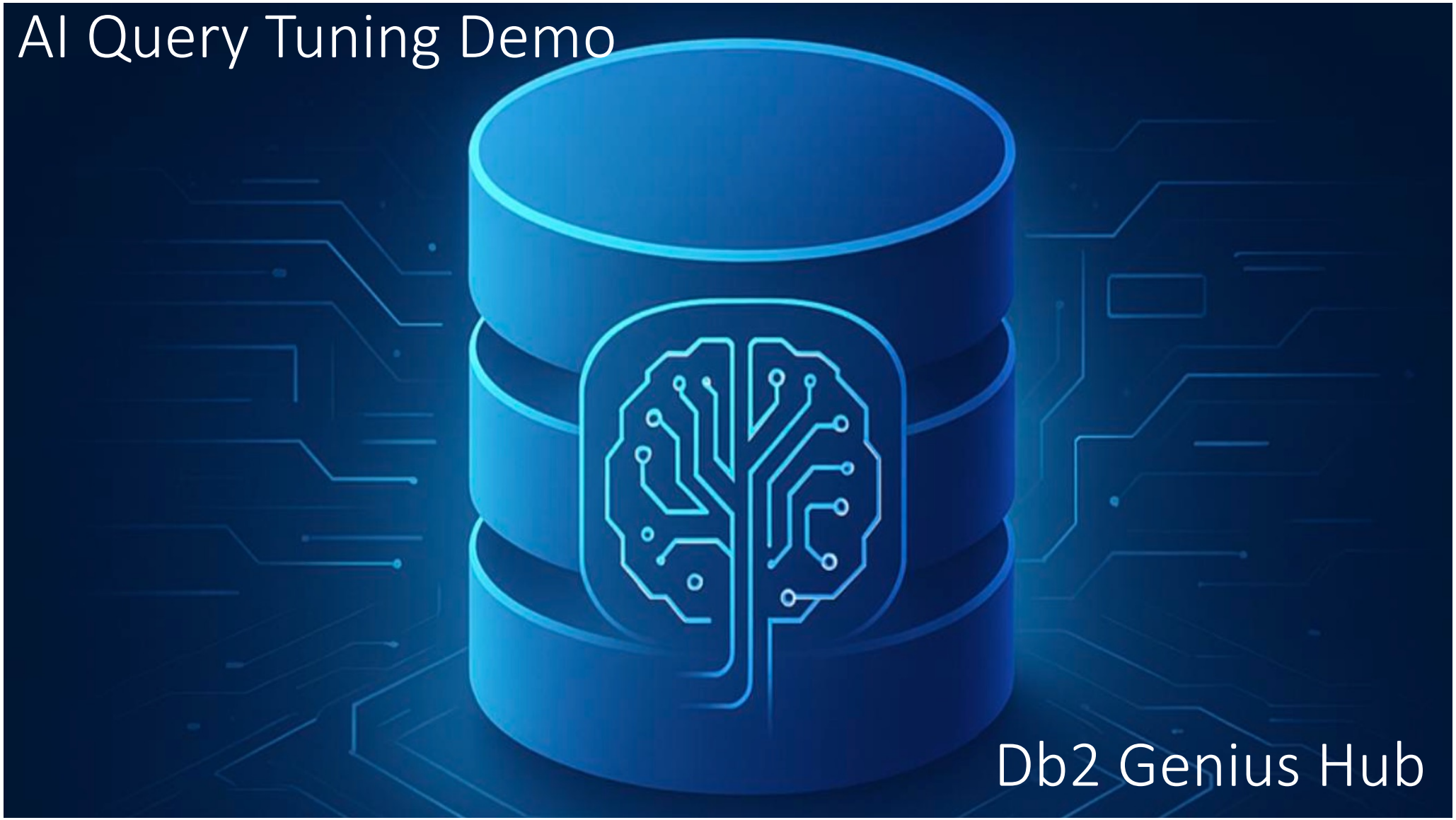
AI: **3** minutes

AI Query Tuning



*coming soon

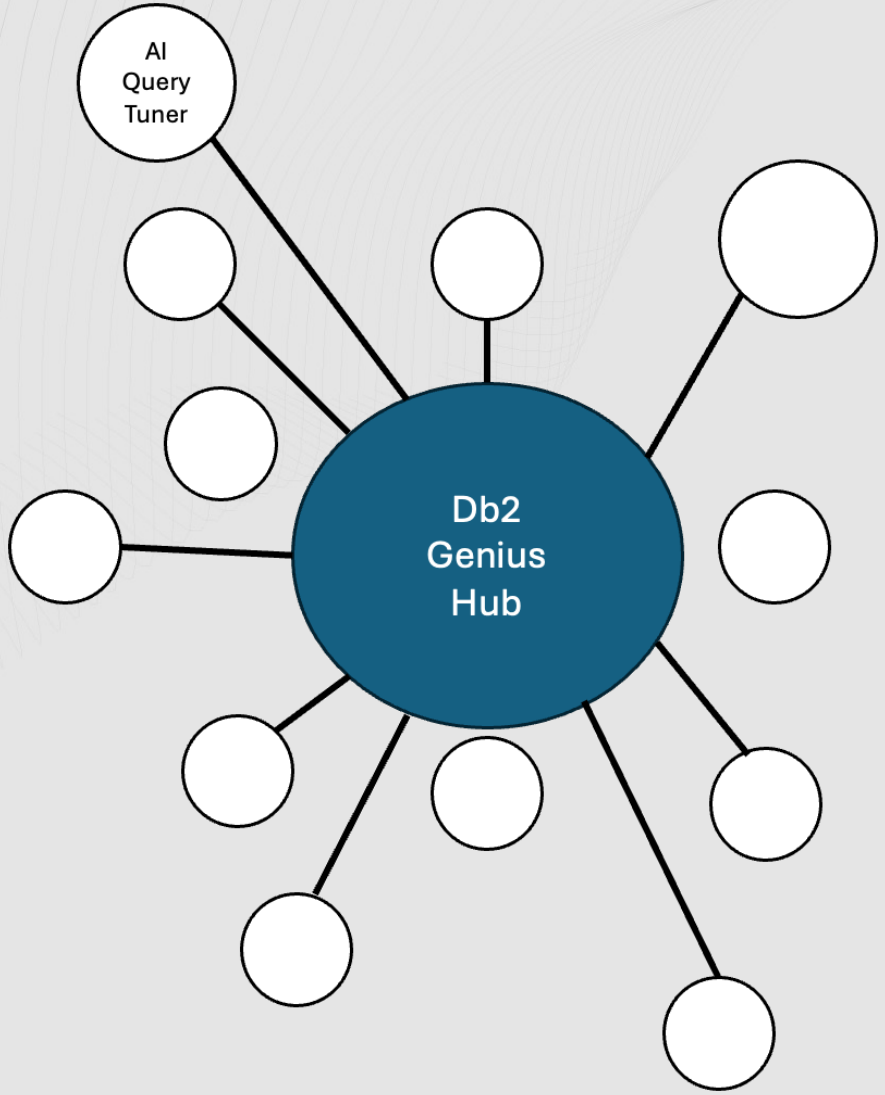
AI Query Tuning Demo



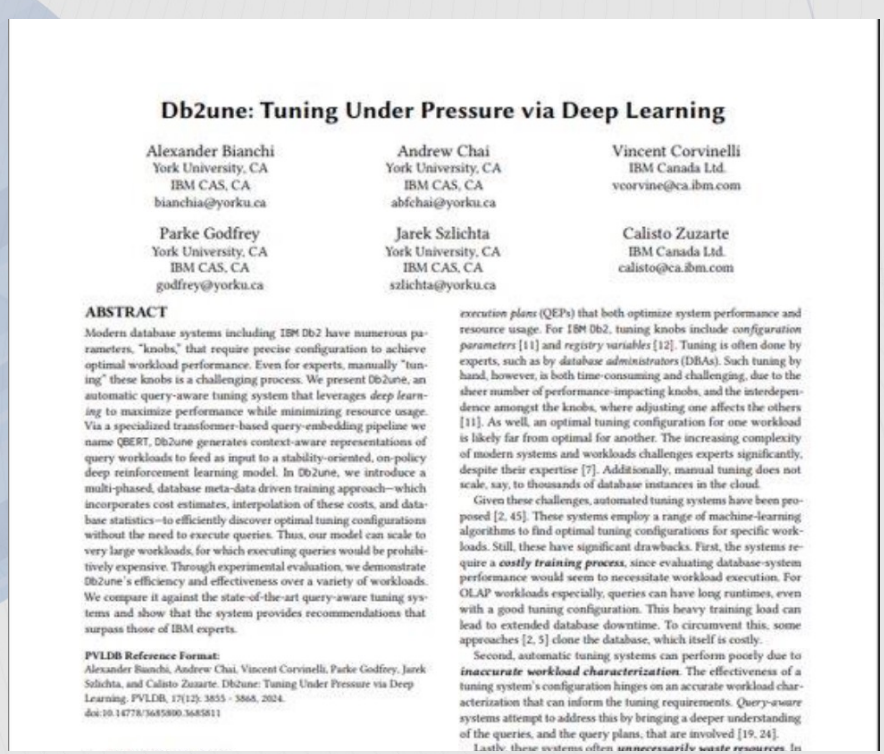
Db2 Genius Hub

Db2 Genius Hub

- Simplify database and query performance tuning
- The Db2 Genius Hub provides insights and smart recommendations through a natural language chat interface.
- The AI Query Tuner provides recommendations for a given query



Database Knob Tuning - Research



Db2une: Tuning Under Pressure via Deep Learning

Alexander Bianchi
York University, CA
IBM CAS, CA
bianchia@yorku.ca

Andrew Chai
York University, CA
IBM CAS, CA
abfchai@yorku.ca

Vincent Corvinelli
IBM Canada Ltd.
vcorvine@ca.ibm.com

Parke Godfrey
York University, CA
IBM CAS, CA
godfrey@yorku.ca

Jarek Szlichta
York University, CA
IBM CAS, CA
szlichta@yorku.ca

Calisto Zuzarte
IBM Canada Ltd.
calisto@ca.ibm.com

ABSTRACT

Modern database systems including IBM Db2 have numerous parameters, "knobs," that require precise configuration to achieve optimal workload performance. Even for experts, manually "tuning" these knobs is a challenging process. We present Db2une, an automatic query-aware tuning system that leverages *deep learning* to maximize performance while minimizing resource usage. Via a specialized transformer-based query-embedding pipeline we name QBERT, Db2une generates context-aware representations of query workloads to feed as input to a stability-oriented, on-policy deep reinforcement learning model. In Db2une, we introduce a multi-phased, database meta-data driven training approach—which incorporates cost estimates, interpolation of these costs, and database statistics—to efficiently discover optimal tuning configurations without the need to execute queries. Thus, our model can scale to very large workloads, for which executing queries would be prohibitively expensive. Through experimental evaluation, we demonstrate Db2une's efficiency and effectiveness over a variety of workloads. We compare it against the state-of-the-art query-aware tuning systems and show that the system provides recommendations that surpass those of IBM experts.

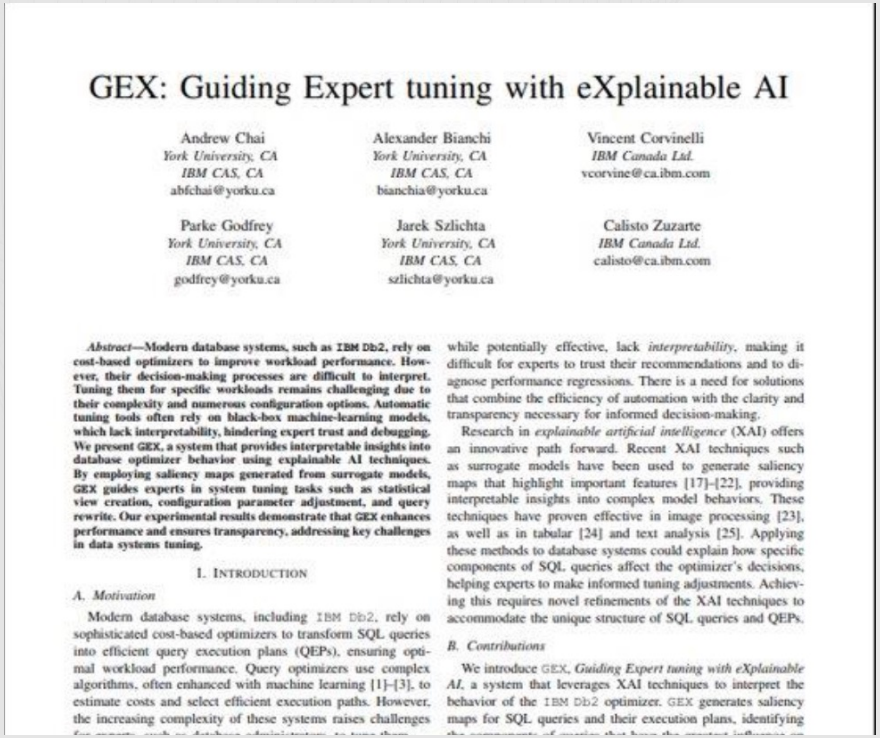
PVLDB Reference Format:
Alexander Bianchi, Andrew Chai, Vincent Corvinelli, Parke Godfrey, Jarek Szlichta, and Calisto Zuzarte. Db2une: Tuning Under Pressure via Deep Learning. PVLDB, 17(12): 3855 - 3868, 2024.
doi:10.14778/3685800.3685811

execution plans (QEPs) that both optimize system performance and resource usage. For IBM Db2, tuning knobs include *configuration parameters* [11] and *registry variables* [12]. Tuning is often done by experts, such as by *database administrators* (DBAs). Such tuning by hand, however, is both time-consuming and challenging, due to the sheer number of performance-impacting knobs, and the interdependence amongst the knobs, where adjusting one affects the others [11]. As well, an optimal tuning configuration for one workload is likely far from optimal for another. The increasing complexity of modern systems and workloads challenges experts significantly, despite their expertise [7]. Additionally, manual tuning does not scale, say, to thousands of database instances in the cloud.

Given these challenges, automated tuning systems have been proposed [2, 45]. These systems employ a range of machine-learning algorithms to find optimal tuning configurations for specific workloads. Still, these have significant drawbacks. First, the systems require a ***costly training process***, since evaluating database-system performance would seem to necessitate workload execution. For OLAP workloads especially, queries can have long runtimes, even with a good tuning configuration. This heavy training load can lead to extended database downtime. To circumvent this, some approaches [2, 5] clone the database, which itself is costly.

Second, automatic tuning systems can perform poorly due to ***inaccurate workload characterization***. The effectiveness of a tuning system's configuration hinges on an accurate workload characterization that can inform the tuning requirements. *Query-aware* systems attempt to address this by bringing a deeper understanding of the queries, and the query plans, that are involved [19, 24].

Lastly, these systems often ***unnecessarily waste resources***. In



GEX: Guiding Expert tuning with eXplainable AI

Andrew Chai
York University, CA
IBM CAS, CA
abfchai@yorku.ca

Alexander Bianchi
York University, CA
IBM CAS, CA
bianchia@yorku.ca

Vincent Corvinelli
IBM Canada Ltd.
vcorvine@ca.ibm.com

Parke Godfrey
York University, CA
IBM CAS, CA
godfrey@yorku.ca

Jarek Szlichta
York University, CA
IBM CAS, CA
szlichta@yorku.ca

Calisto Zuzarte
IBM Canada Ltd.
calisto@ca.ibm.com

***Abstract*—Modern database systems, such as IBM Db2, rely on cost-based optimizers to improve workload performance. However, their decision-making processes are difficult to interpret. Tuning them for specific workloads remains challenging due to their complexity and numerous configuration options. Automatic tuning tools often rely on black-box machine-learning models, which lack interpretability, hindering expert trust and debugging. We present GEX, a system that provides interpretable insights into database optimizer behavior using explainable AI techniques. By employing saliency maps generated from surrogate models, GEX guides experts in system tuning tasks such as statistical view creation, configuration parameter adjustment, and query rewrite. Our experimental results demonstrate that GEX enhances performance and ensures transparency, addressing key challenges in data systems tuning.**

I. Introduction

A. Motivation

Modern database systems, including IBM Db2, rely on sophisticated cost-based optimizers to transform SQL queries into efficient query execution plans (QEPs), ensuring optimal workload performance. Query optimizers use complex algorithms, often enhanced with machine learning [1]–[3], to estimate costs and select efficient execution paths. However, the increasing complexity of these systems raises challenges

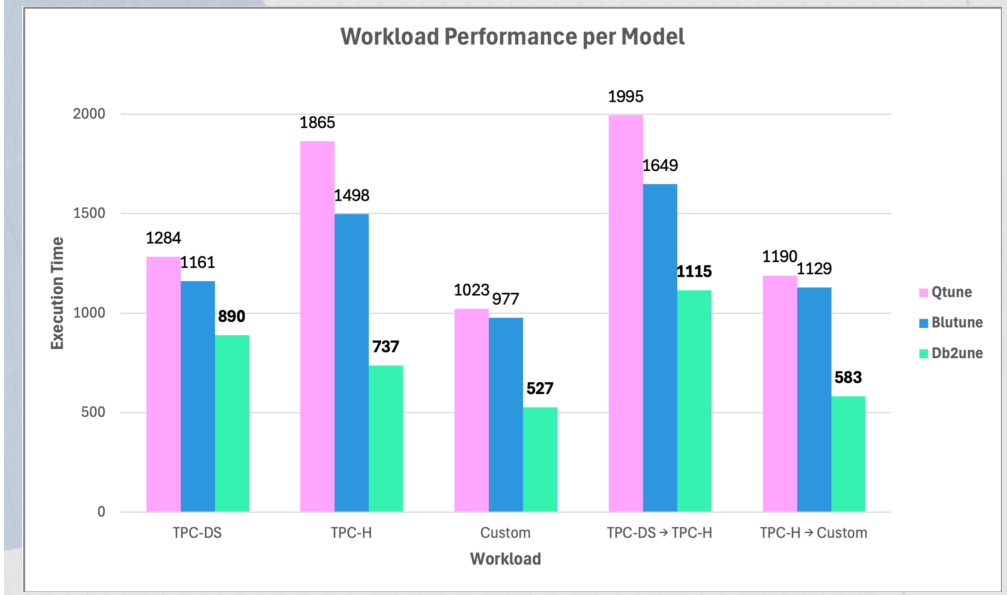
while potentially effective, lack *interpretability*, making it difficult for experts to trust their recommendations and to diagnose performance regressions. There is a need for solutions that combine the efficiency of automation with the clarity and transparency necessary for informed decision-making.

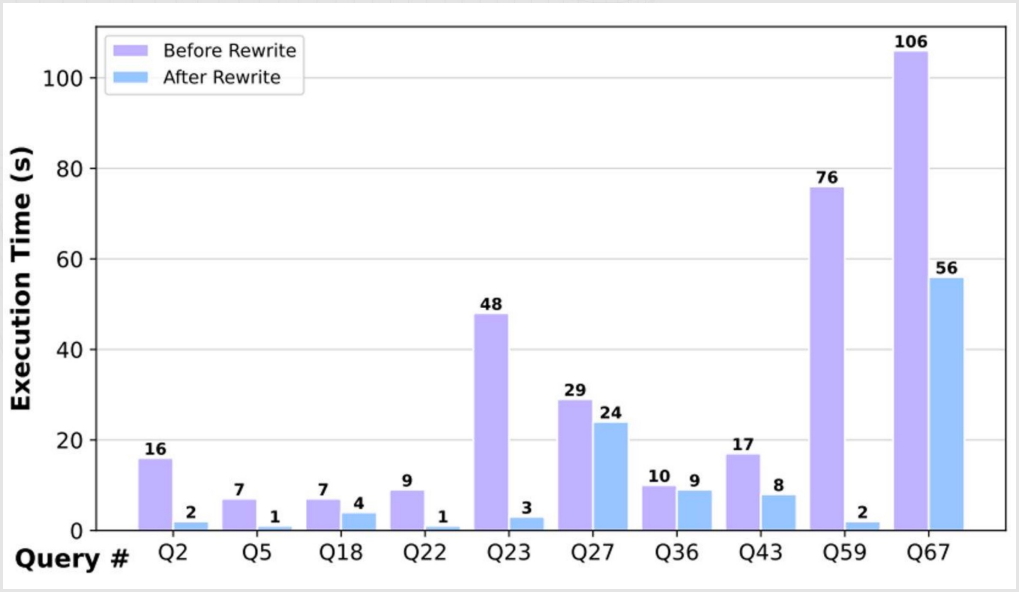
Research in *explainable artificial intelligence* (XAI) offers an innovative path forward. Recent XAI techniques such as surrogate models have been used to generate saliency maps that highlight important features [17]–[22], providing interpretable insights into complex model behaviors. These techniques have proven effective in image processing [23], as well as in tabular [24] and text analysis [25]. Applying these methods to database systems could explain how specific components of SQL queries affect the optimizer's decisions, helping experts to make informed tuning adjustments. Achieving this requires novel refinements of the XAI techniques to accommodate the unique structure of SQL queries and QEPs.

B. Contributions

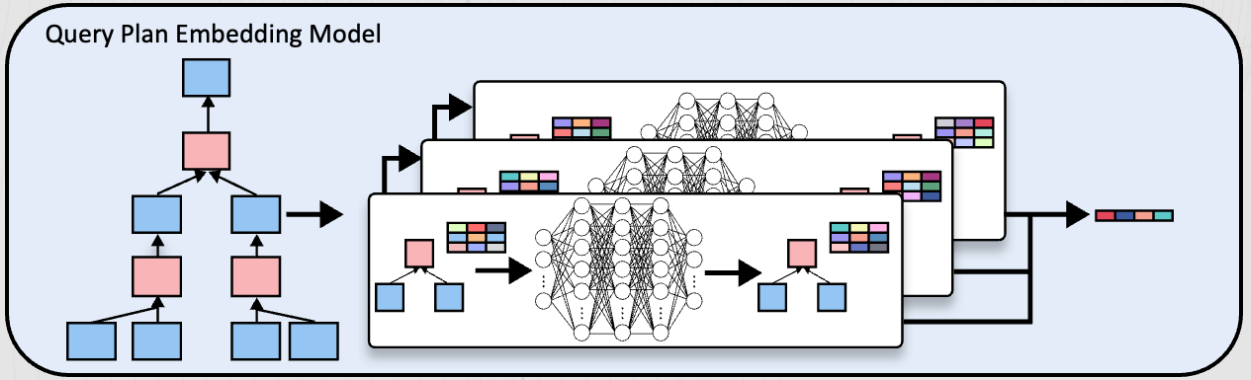
We introduce GEX, *Guiding Expert tuning with eXplainable AI*, a system that leverages XAI techniques to interpret the behavior of the IBM Db2 optimizer. GEX generates saliency maps for SQL queries and their execution plans, identifying

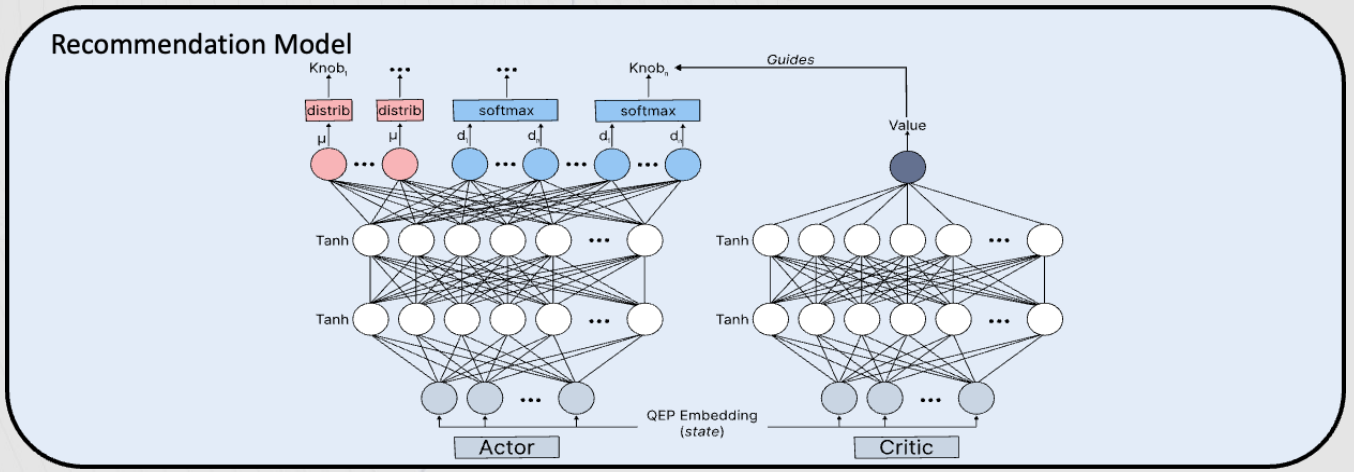
Research Prototype - Experimental Results





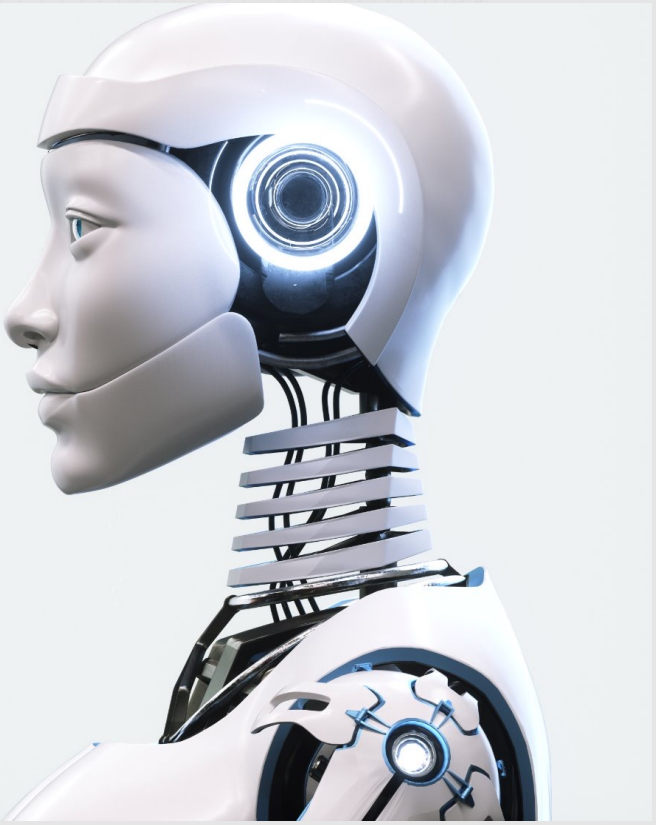
AI Query Tuner Model





AI Query Optimizer: Cardinality Estimation

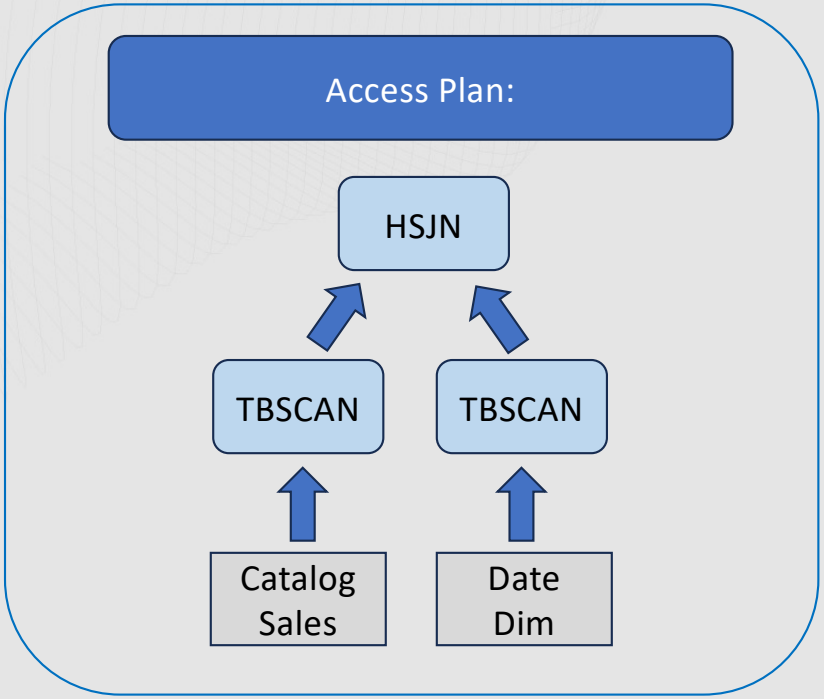
- Review of Optimization
- Cardinality Estimation
- AI Implementation
- Performance Improvements
- Savings
- Future



Overview of Query Optimization

Query Optimization

- Generates alternative access plans
- Estimates cost of each plan
- Selects the cheapest plan
- Depends heavily on cardinality estimates



Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = ‘NA’ and O.orderdate >= '2022-01-01’
```

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = ‘NA’ and O.orderdate >= '2022-01-01’
```

= cardinalilty of Customers

x cardinality of Orders

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = ‘NA’ and O.orderdate >= '2022-01-01’
```

= 1000

x 54,995

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x selectivity of C.region = 'NA'

x 54,995 x selectivity of O.orderdate >= '2022-01-01'

x selectivity of C.custid = O.custid

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x selectivity of C.region = 'NA'

x 54,995 x selectivity of O.orderdate >= '2022-01-01'

x selectivity of C.custid = O.custid

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / (# distinct value in c.region)

x 54,995 x selectivity of O.orderdate >= '2022-01-01'

x selectivity of C.custid = O.custid

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x selectivity of O.orderdate >= '2022-01-01'

x selectivity of C.custid = O.custid

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x selectivity of O.orderdate >= '2022-01-01'

x selectivity of C.custid = O.custid

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x (max – '2022-01-01') / (max – min)

x selectivity of C.custid = O.custid

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x ('2024-12-31' – '2022-01-01)

/ ('2024-12-31' – '2015-01-01')

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x 3 years

/ 10 years

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x 0.3

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x 0.3

x selectivity of C.custid = O.custid

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x 0.3

x 1 / max(#distinct values in C.custid, #distinct values in O.custid)

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2

x 54,995 x 0.3

x 1 / max(1000,850)

Traditional Join Cardinality Estimation

SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = ‘NA’ and O.orderdate >= '2022-01-01’

= 1000 x 1 / 2
x 54,995 x 0.3
x 1 / max(1000,850)
= 8249

Traditional Join Cardinality Estimation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

= 1000 x 1 / 2
x 54,995 x 0.3
x 1 / max(1000,850)

= 8249 vs actual of 25,524 = error of ~68%

Question: Why?

Answer:

Non-uniformity and Correlation

```
SELECT *
FROM Customers C JOIN Orders O ON C.custid = O.custid
WHERE C.region = 'NA' and O.orderdate >= '2022-01-01'
```

- Non-uniformity
  - Higher percentage of customers in North America
  - Increase in sales over time
- Correlation
  - Increase of sales in North America disproportionate to other regions

AI Estimate:

= Can AI do better?

25,859 (error = ~1%)

Tuning for Good Cardinality Estimates

Default Statistics

```
Est:          1,137
Actual: 10,113,972
        HSJOIN
     /----+----\
2,000,000      |
CUSTOMER    Est: 1137
            HSJOIN
         /-----+-----\
    2.9E+8      Est: 0.29
  STORE_SALES    DATE_DIM
```

Column Group Statistics

```
Est:        457,723
Actual: 10,113,972
        HSJOIN
     /----+----\
2,000,000      |
CUSTOMER    Est: 457,723
            HSJOIN
         /-----+-----\
    2.9E+8      Est: 116
  STORE_SALES    DATE_DIM
```

Statistical Views

```
Est:      9,053,830
Actual: 10,113,972
          HSJOIN
       /------+------\
       |             2,000,000
Est: 9,053,830       CUSTOMER
    HSJOIN
 /----+------\
2.9E+8        Est: 116
STORE_SALES   DATE_DIM
```

Question: Can AI do better? Or at least save valuable tuning time?

How does AI estimate cardinality?

- single table cardinality model available in version 12 GA
- Trained
  - during auto runstats
  - using ~1,000,000 automatically generated queries based on runstats sample
- Queries are represented
  - by the min and max for each column after the application of local predicates

```
Select * from orders
where custid = 2
 and profit >= 91.00
```

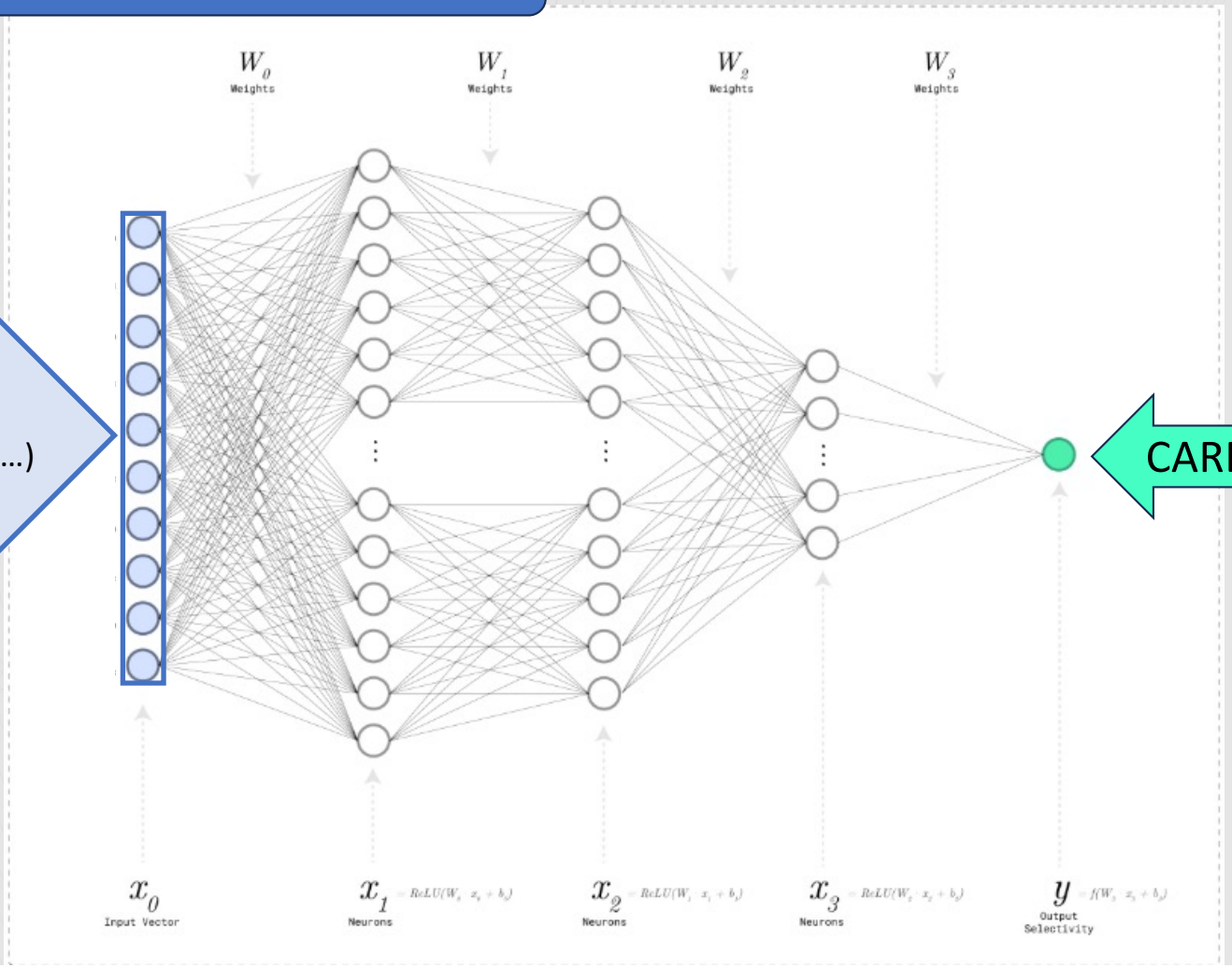
orderid	custid	profit
1	3	98.00
2	2	65.00
3	2	91.00
4	1	70.00

Features:	(0.0 , 1.0	0.25, 0.75	0.5, 1.0

AI Uses Neural Networks

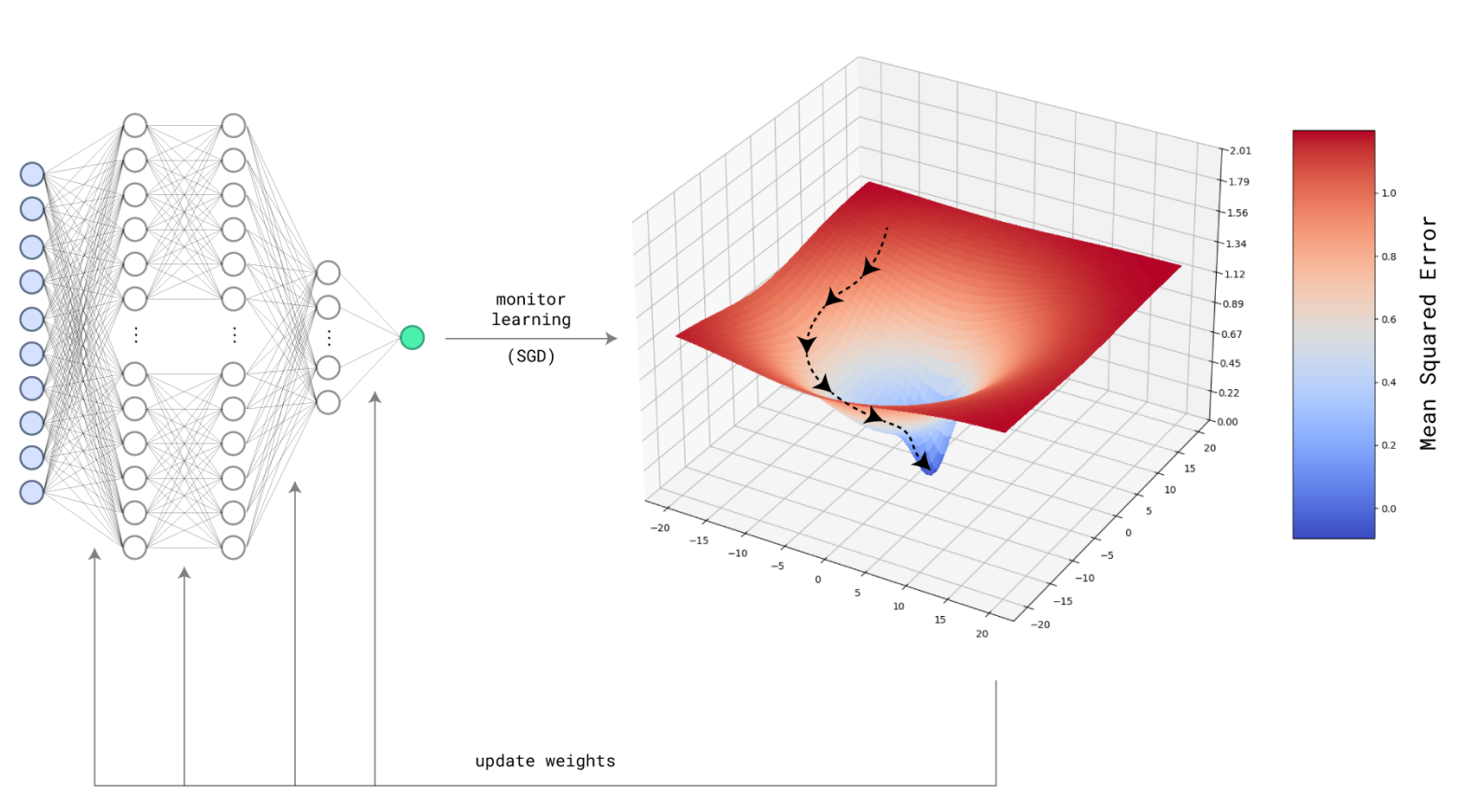
C1 in

FEATURE VECTOR
(c1min, c1max, c2min, c2max,…)



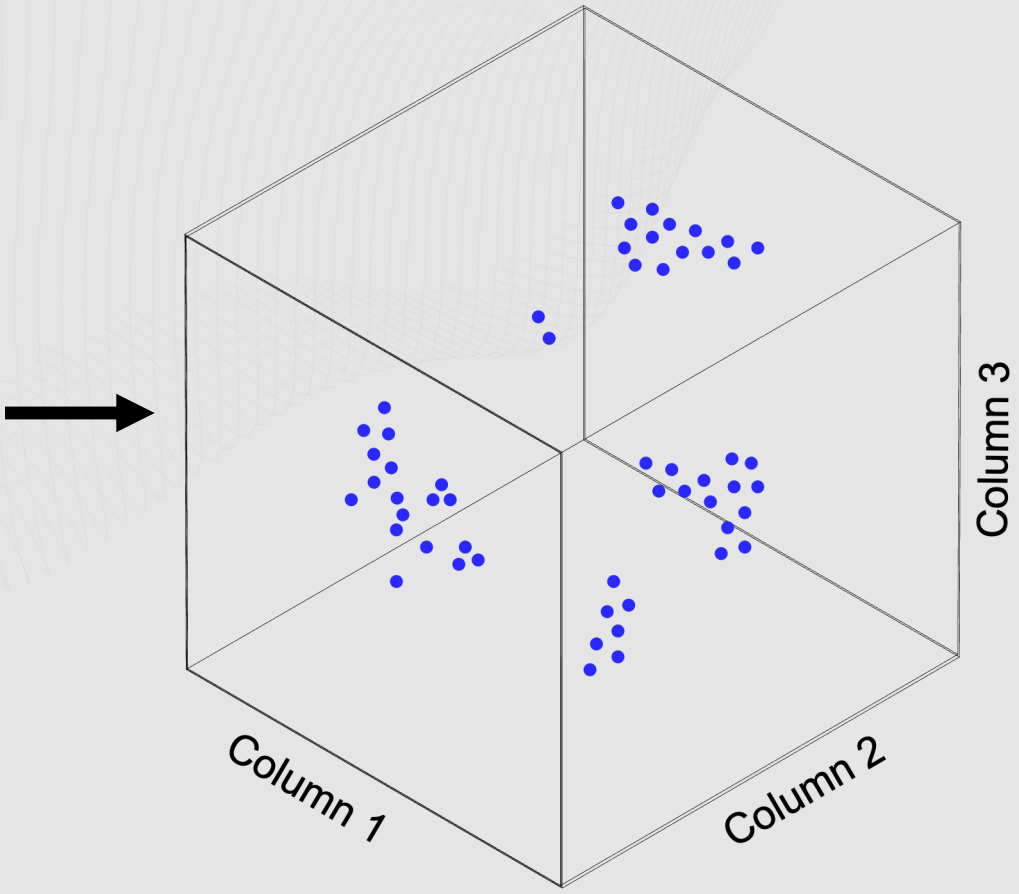
CARDINALITY

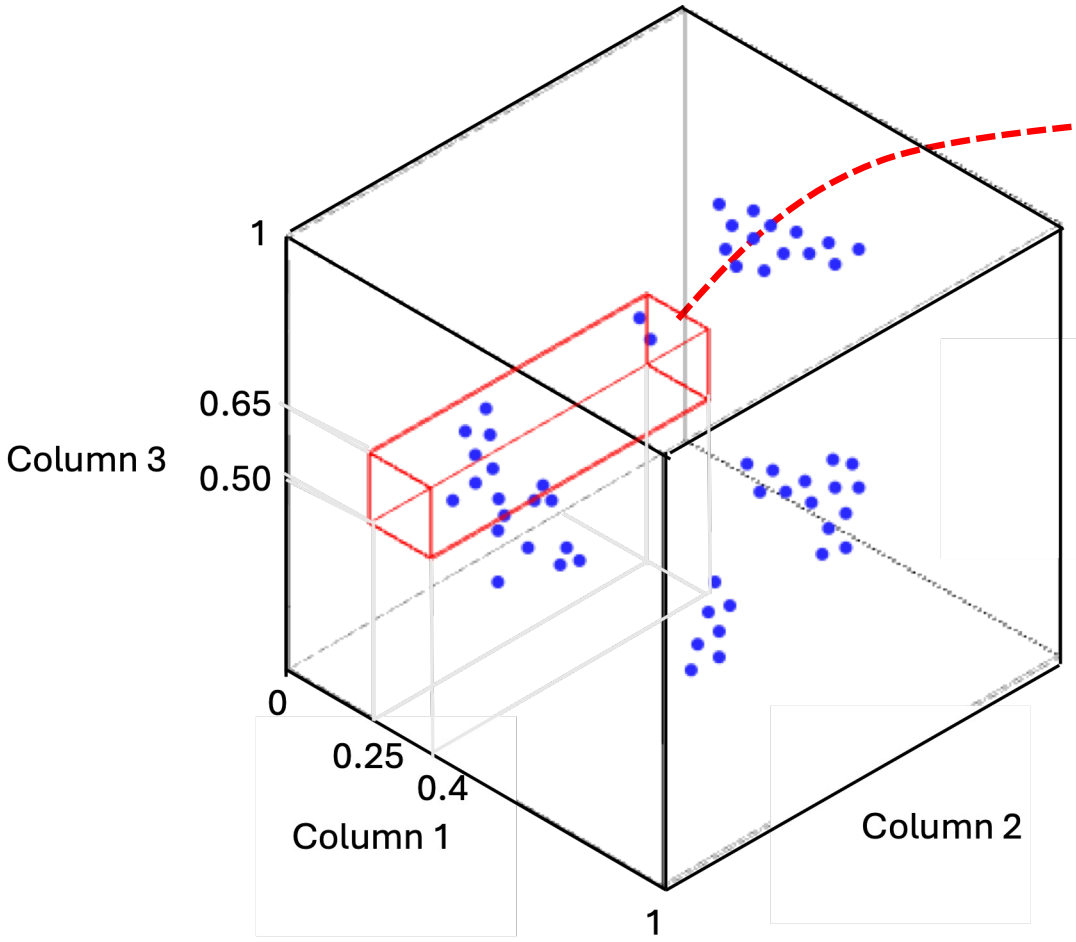
Trained Using Gradient Descent



Intuitively:

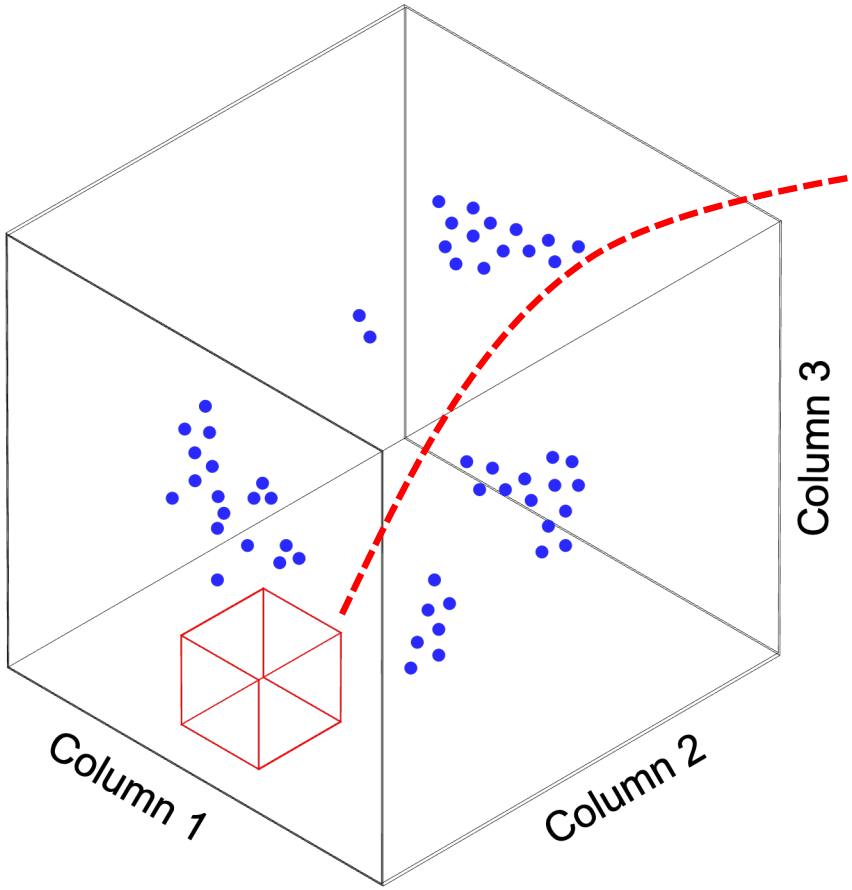
Column 1	Column 2	Column 3
4	‘Canada’	15
7	‘Japan’	17
7	‘Canada’	88
3	‘Canada’	15
…	…	…





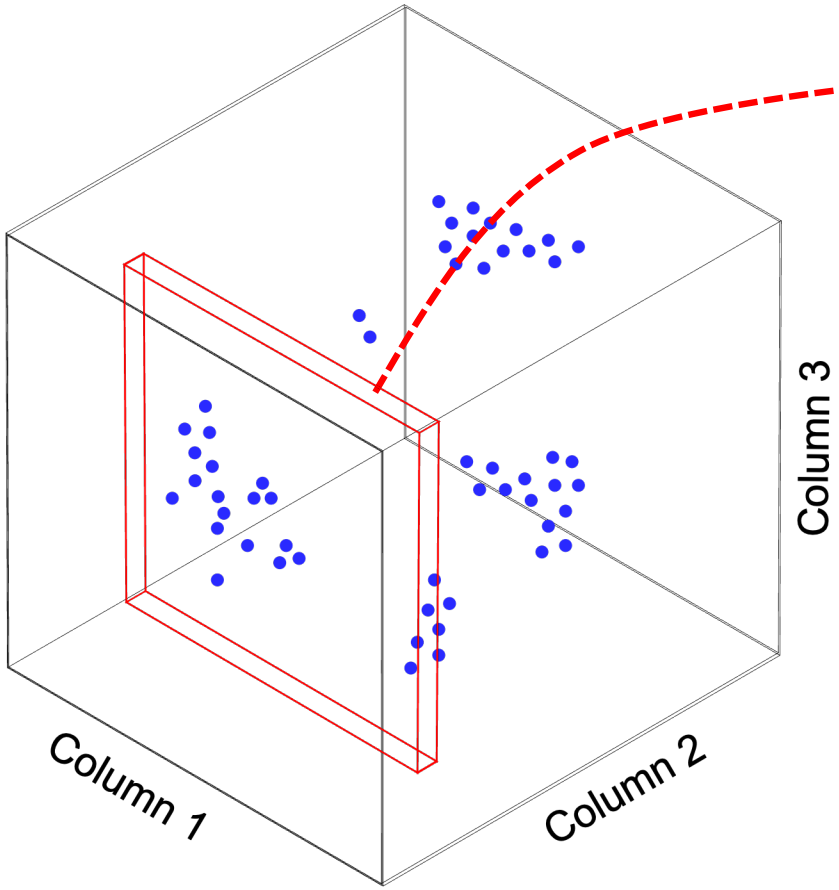
15%

```
SELECT
     COUNT(*)
FROM
     T1
WHERE
     COLUMN 1 BETWEEN 0.25 AND 0.40
     COLUMN 2 BETWEEN 0.50 AND 0.65
     COLUMN 3 BETWEEN 0.0 and 0.66
```



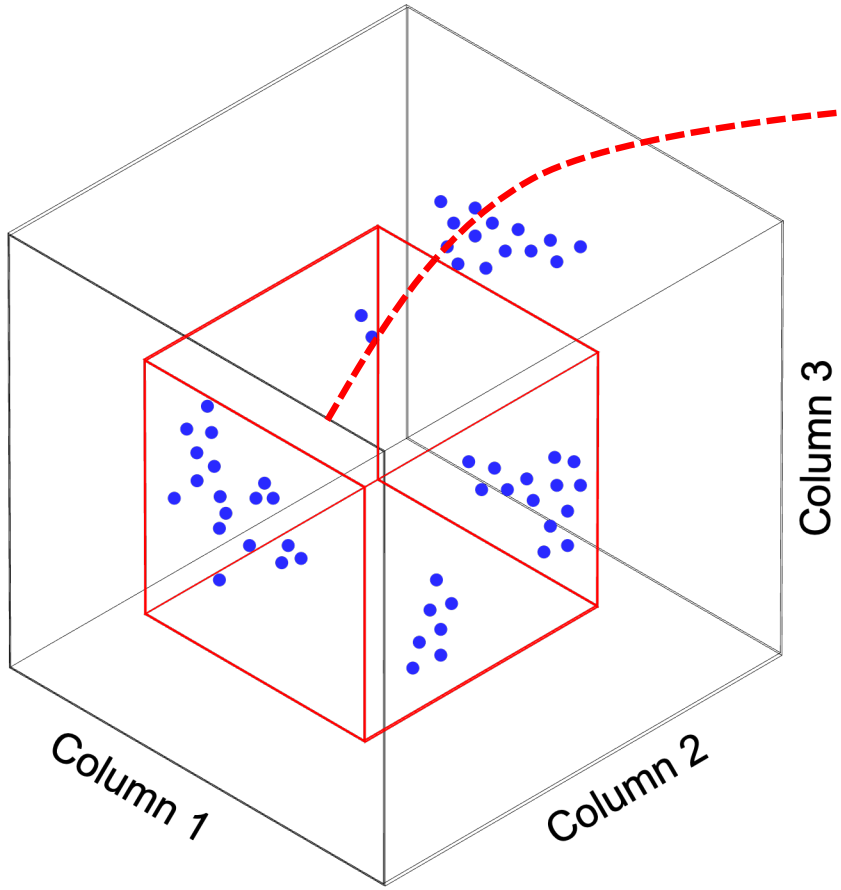
0%

```
SELECT
    COUNT(*)
FROM
    T1
WHERE
    COLUMN 1 BETWEEN 0.5 AND 0.7
    COLUMN 2 BETWEEN 0.1 AND 0.3
    COLUMN 3 BETWEEN 0.0 and 0.2
```



10%

```
SELECT
     COUNT(*)
FROM
     T1
WHERE
     COLUMN 1 BETWEEN 0.0 AND 0.80
     COLUMN 2 BETWEEN 0.20 AND 0.25
     COLUMN 3 BETWEEN 0.0 and 0.80
```

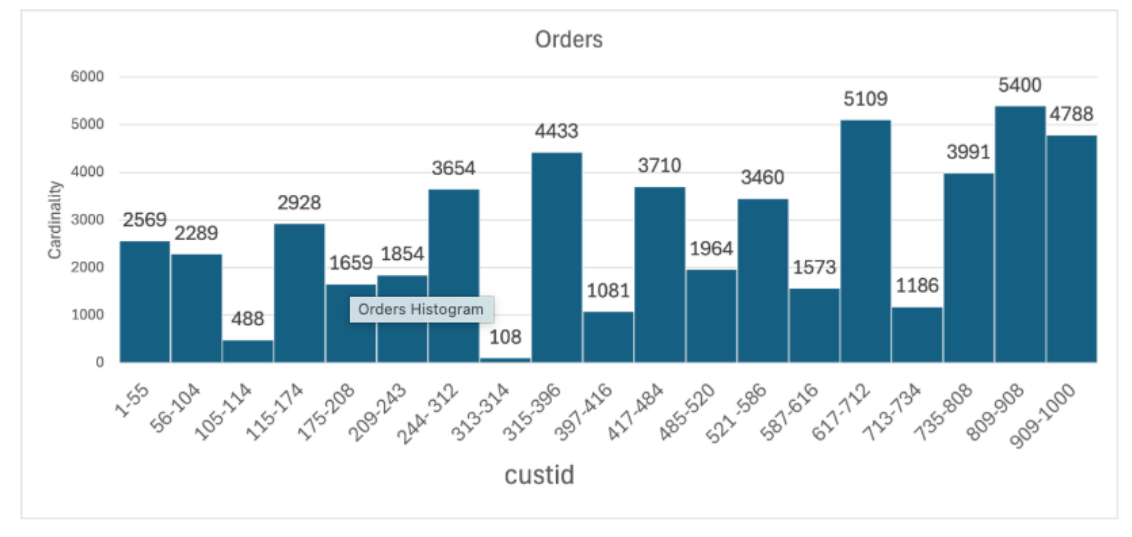


82%

```
SELECT
    COUNT(*)
FROM
    T1
WHERE
    COLUMN 1 BETWEEN 0.05 AND 0.70
    COLUMN 2 BETWEEN 0.25 AND 0.85
    COLUMN 3 BETWEEN 0.90 AND 0.60
```

AI Join Cardinality Estimates

- use single table cardinality models to populate histogram



Select * from orders
where orderdate >= '2022-01-01'
and custid between 1 and 55

AI Join Cardinality Estimates

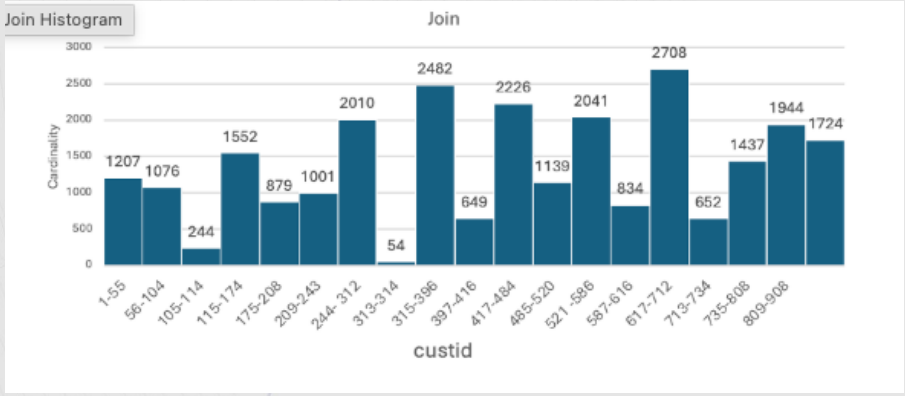
AI Join Cardinality Estimates

∩

AI Join Cardinality Estimates

∩

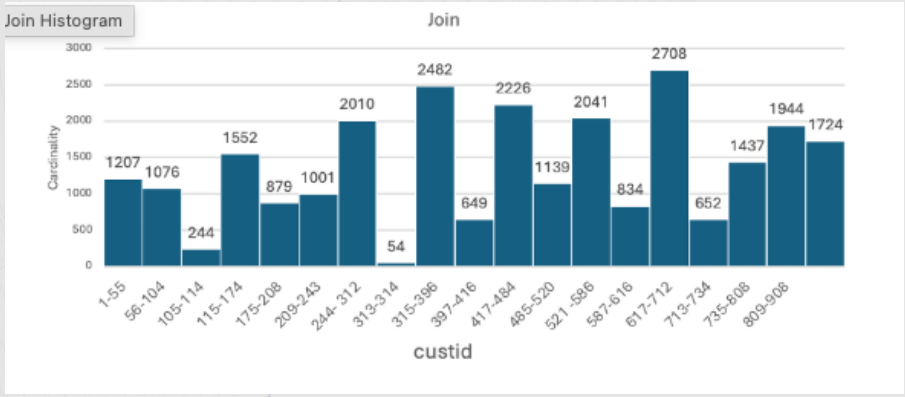
=



AI Join Cardinality Estimates

∩

=

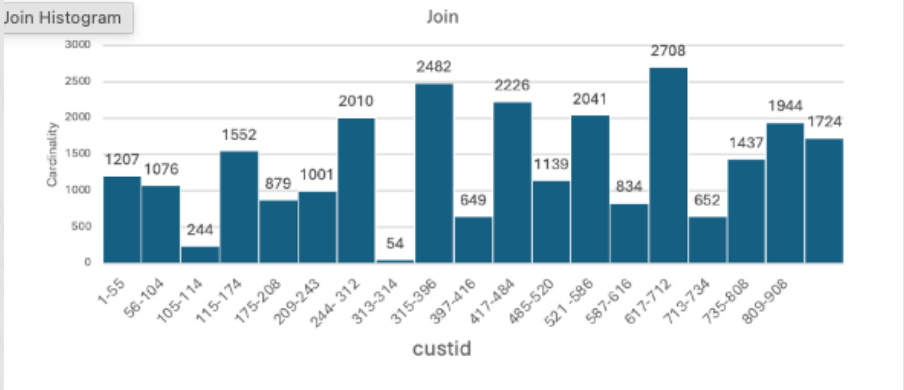


Σ

AI Join Cardinality Estimates

∩

=



$\sum$ = cardinality

Question: Can AI do better? Or at least save valuable tuning time?

Improved Cardinality Estimates

↓

Better Plans

↓

Superior Performace

↓

Less Tuning

↓

Cost Savings

Show me the Money!

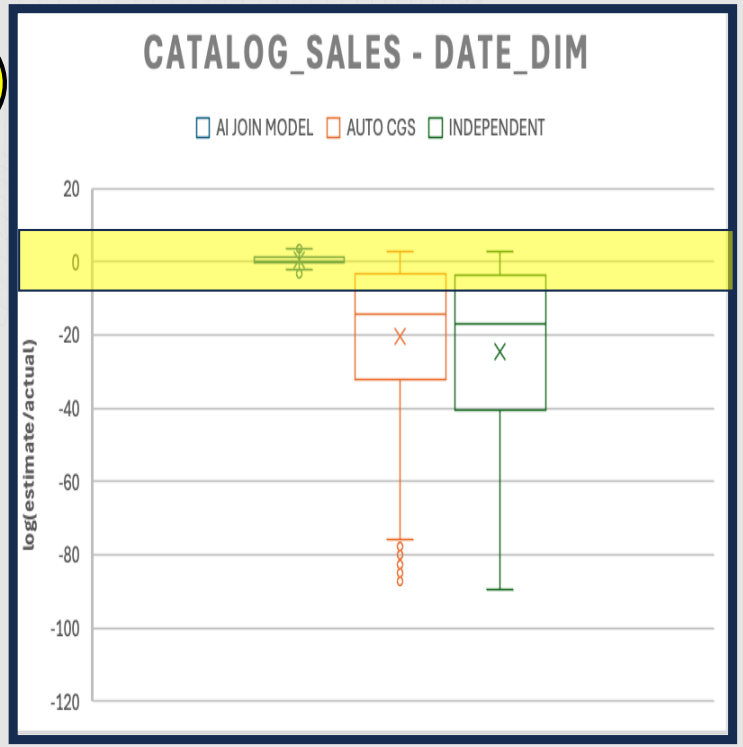
Cardinality Estimate Improvements

1000s queries based on TPCDS schema

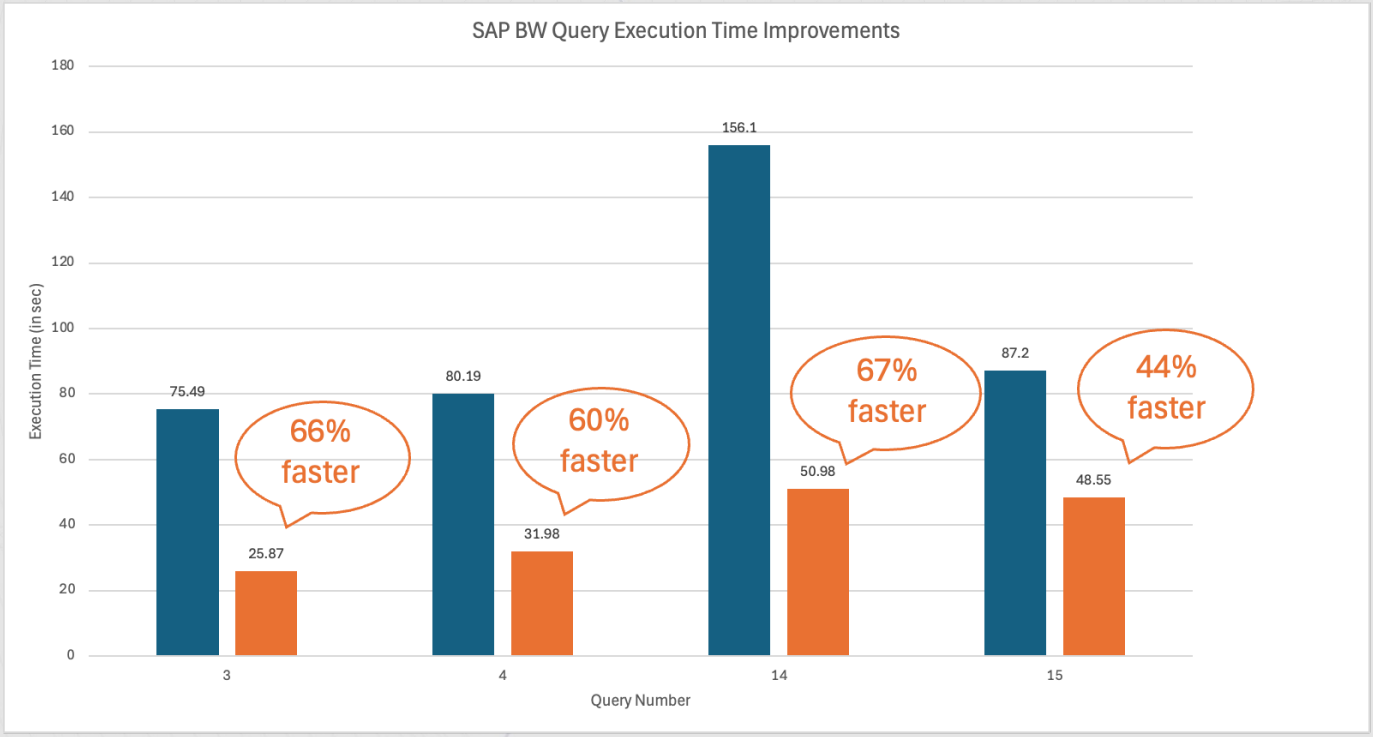
Randomly generated

1-5 local predicates

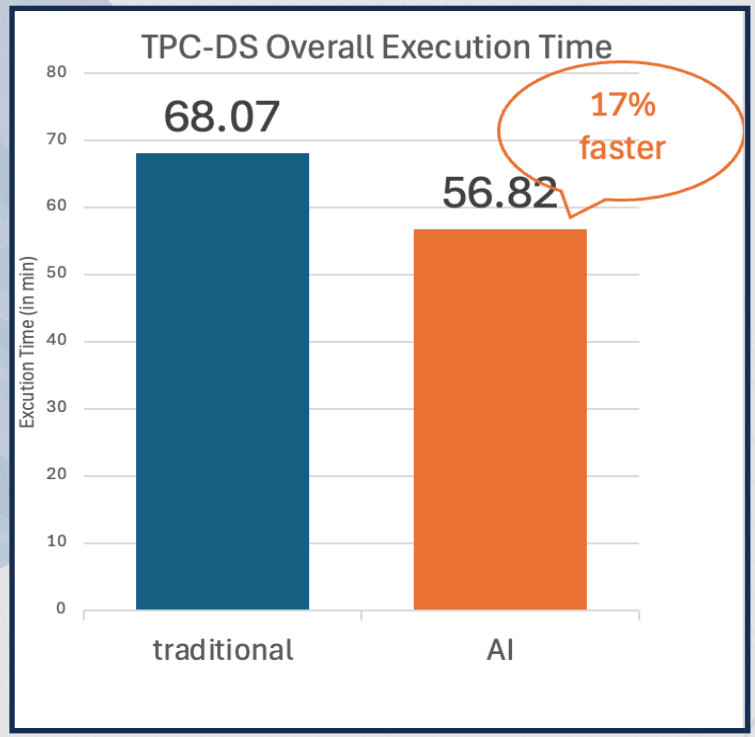
Same join predicates

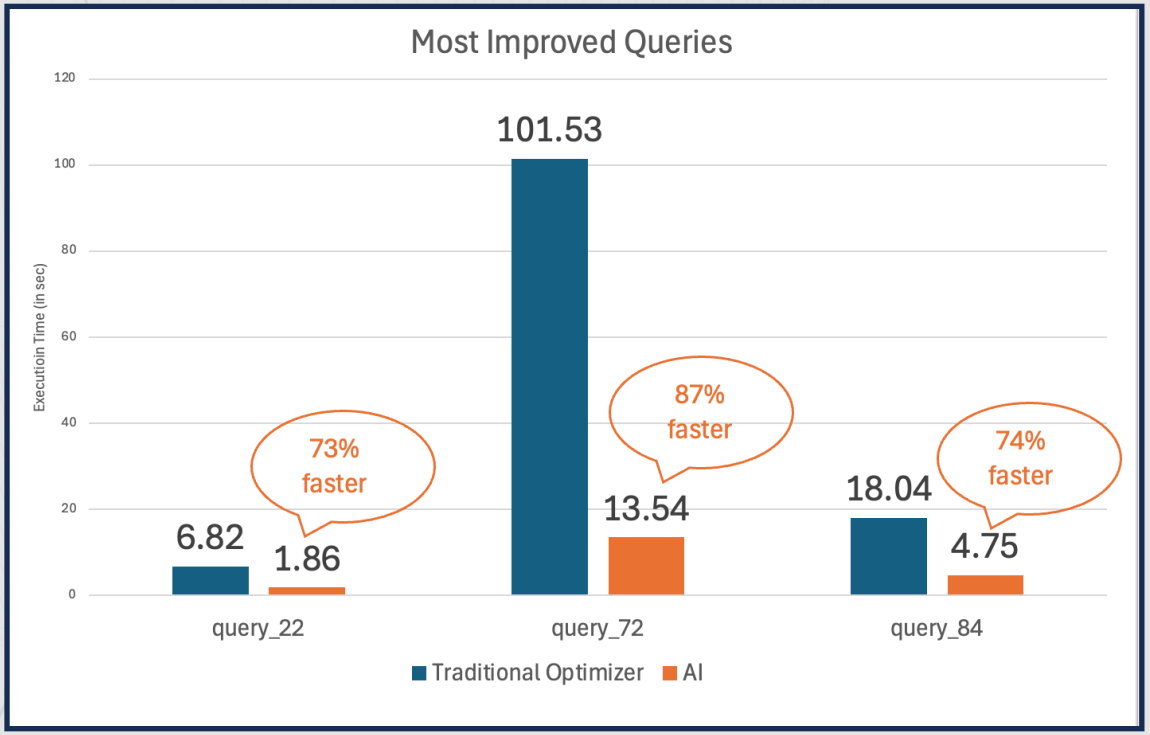


Improved Performance – SAP BW



Improved Performance – TPC-DS

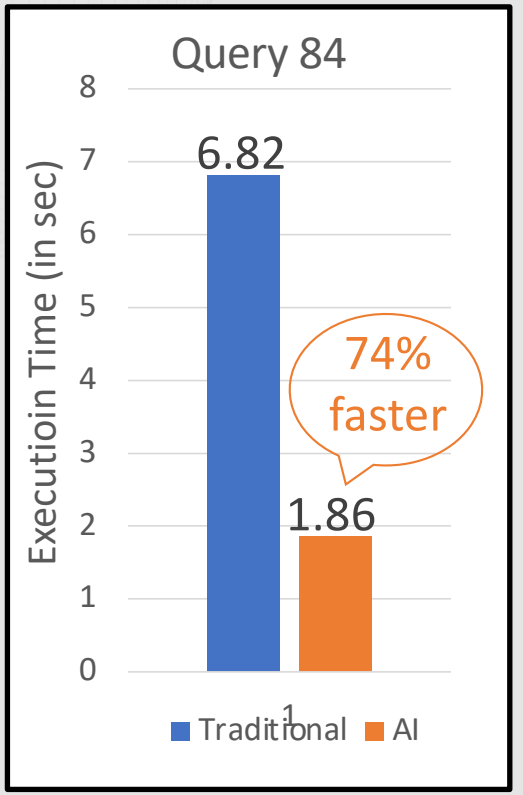




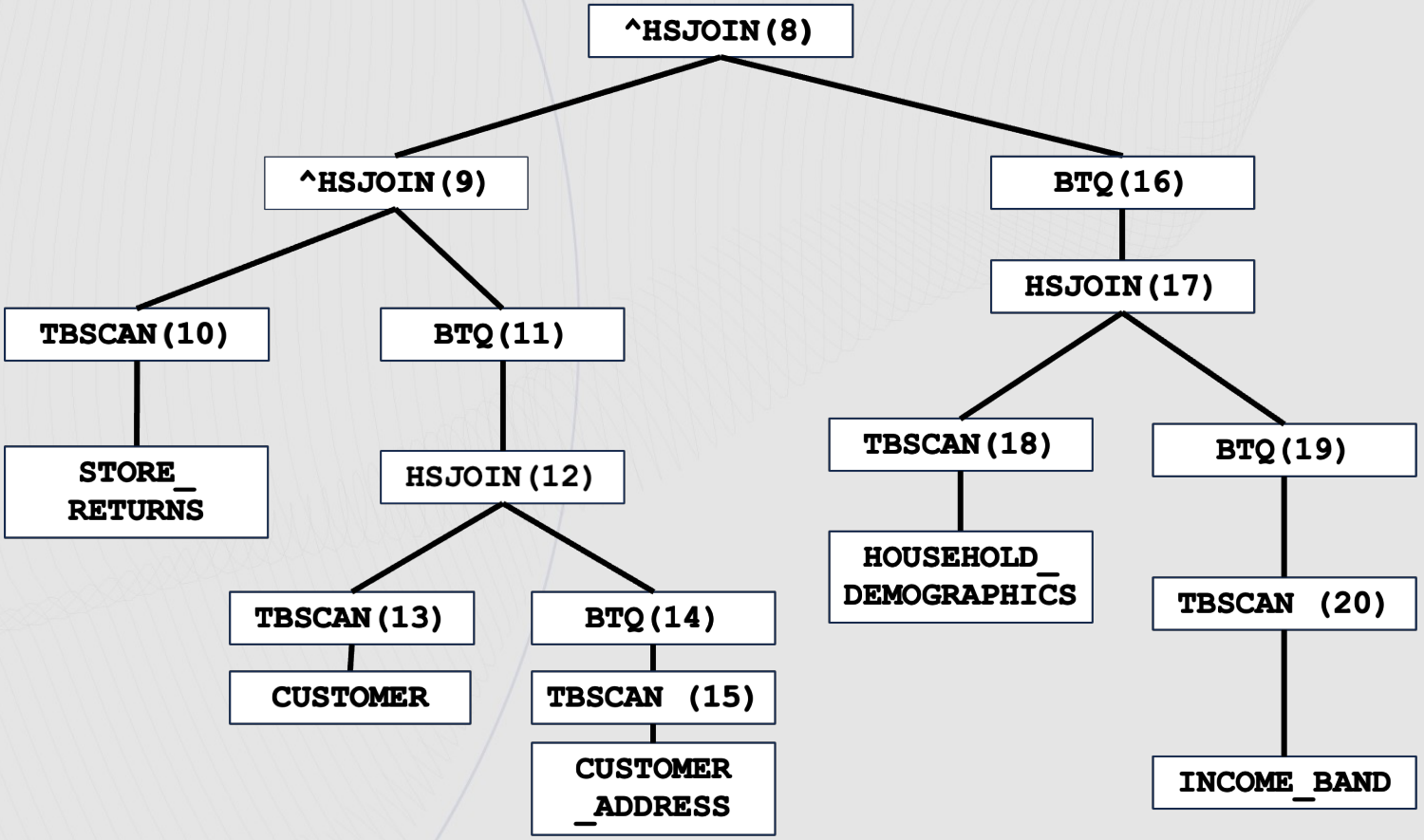
Improved Cardinality, Plans and Performance

TPC-DS Query 84

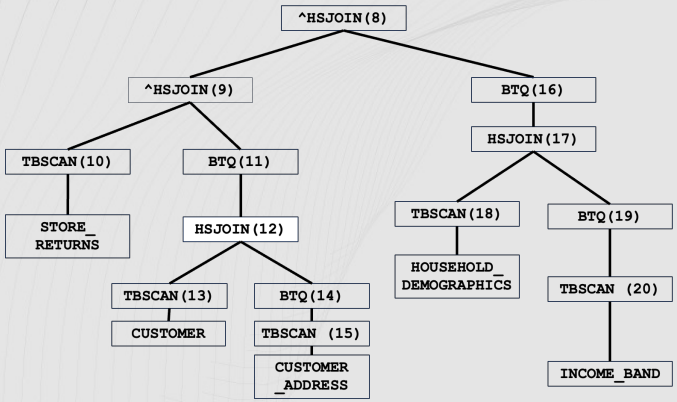
```
select c_customer_id,c_last_name,c_first_name
from customer, customer_address,
     customer_demographics, store_returns,
     household_demographics,income_band
where c_current_addr_sk = ca_address_sk and
      ca_city = 'Greenfield' and
      c_current_cdemo_sk = cd_demo_sk and
      cd_demo_sk = sr_cdemo_sk and
      c_current_hdemo_sk = hd_demo_sk and
      hd_income_band_sk = ib_income_band_sk and
      ib_lower_bound >= 11660 and
      ib_upper_bound <= 11660 + 50000
```



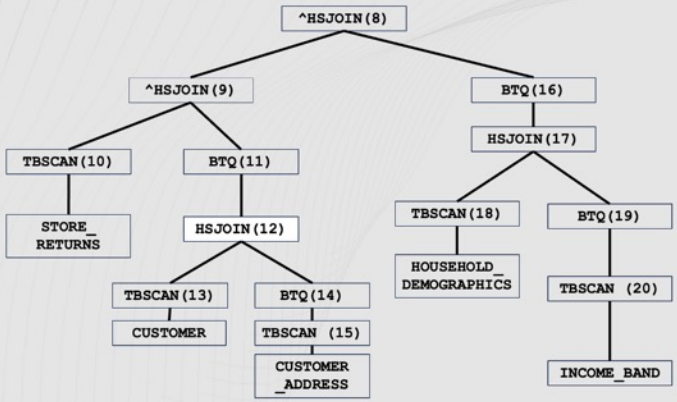
Traditional Plan



Traditional Plan



Traditional Plan



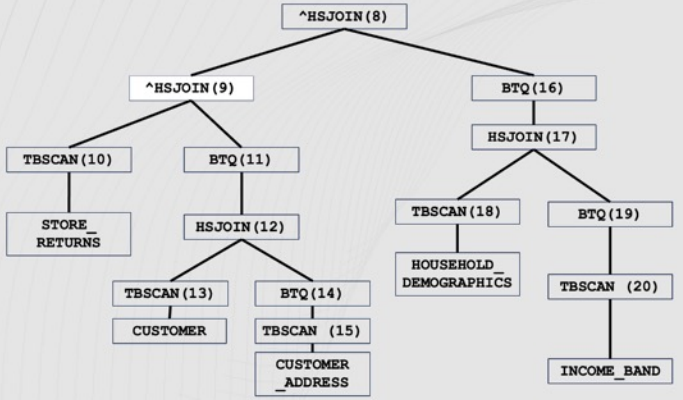
HSJOIN(12)

```
Card(est): trad =  3,196
           AI   = 22,721
Card(actual)    = 30,223
```

← trad card est 10% of actual

← AI card est 75% of actual

Traditional Plan



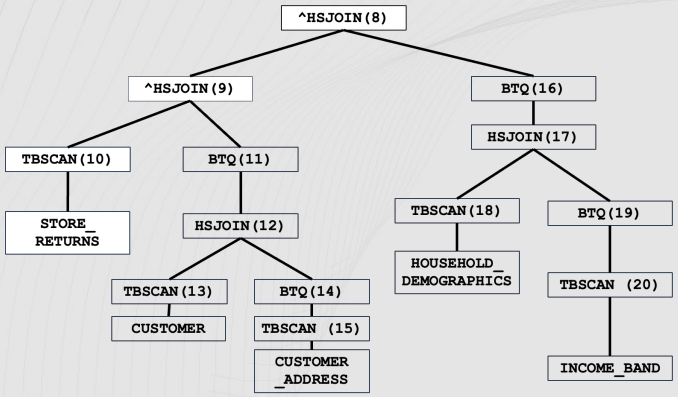
^HSJOIN(9)

```
Cardinality(card)
   estimate(est): trad   = 1.39e+06
                  AI     = 9.88e+06
   actual                = 1.3e+07
```

← trad card est 10% of actual

← AI card est 75% of actual

Traditional Plan



^HSJOIN(8)

```
Cost:
   HSJOIN(8)      : trad = 7.056 vs AI =      50
   OPERATORS 8-20: trad = 36270 vs AI = 36,362
```

AI cost ~10x trad

^HSJOIN(9)

```
Cardinality(card)
   estimate(est): traditional(trad)   = 1.39e+06
                  AI                  = 9.88e+06
   actual                             = 1.3e+07
Cost HSJOIN(9): Trad = 434.01 vs AI = 486.4
```

Poor trad card estimate

Better AI card estimate

Second large HSJOIN probe

cost of join

TBSCAN(10)

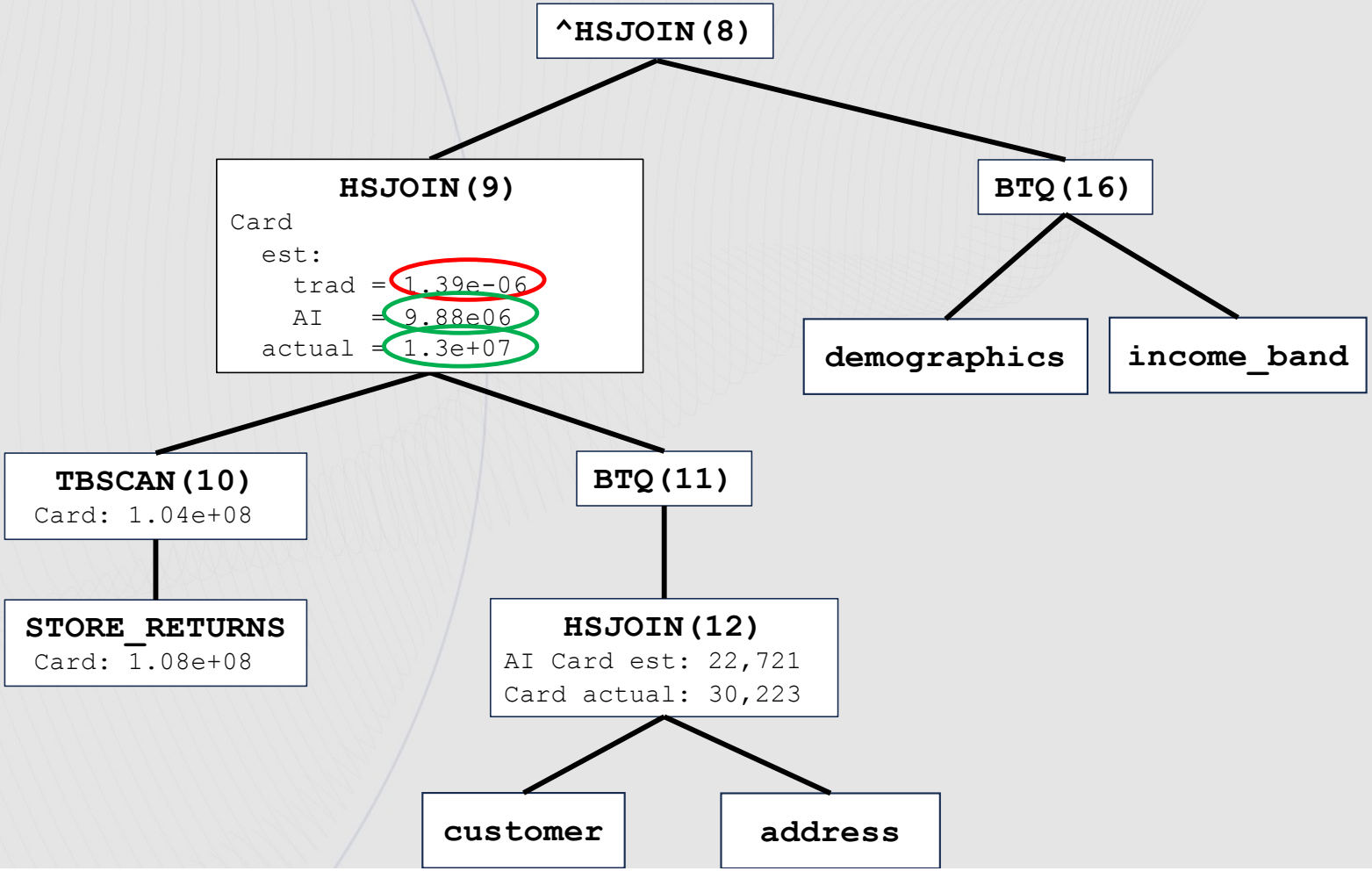
Card(est): 1.04e+08

← First large HSJOIN probe

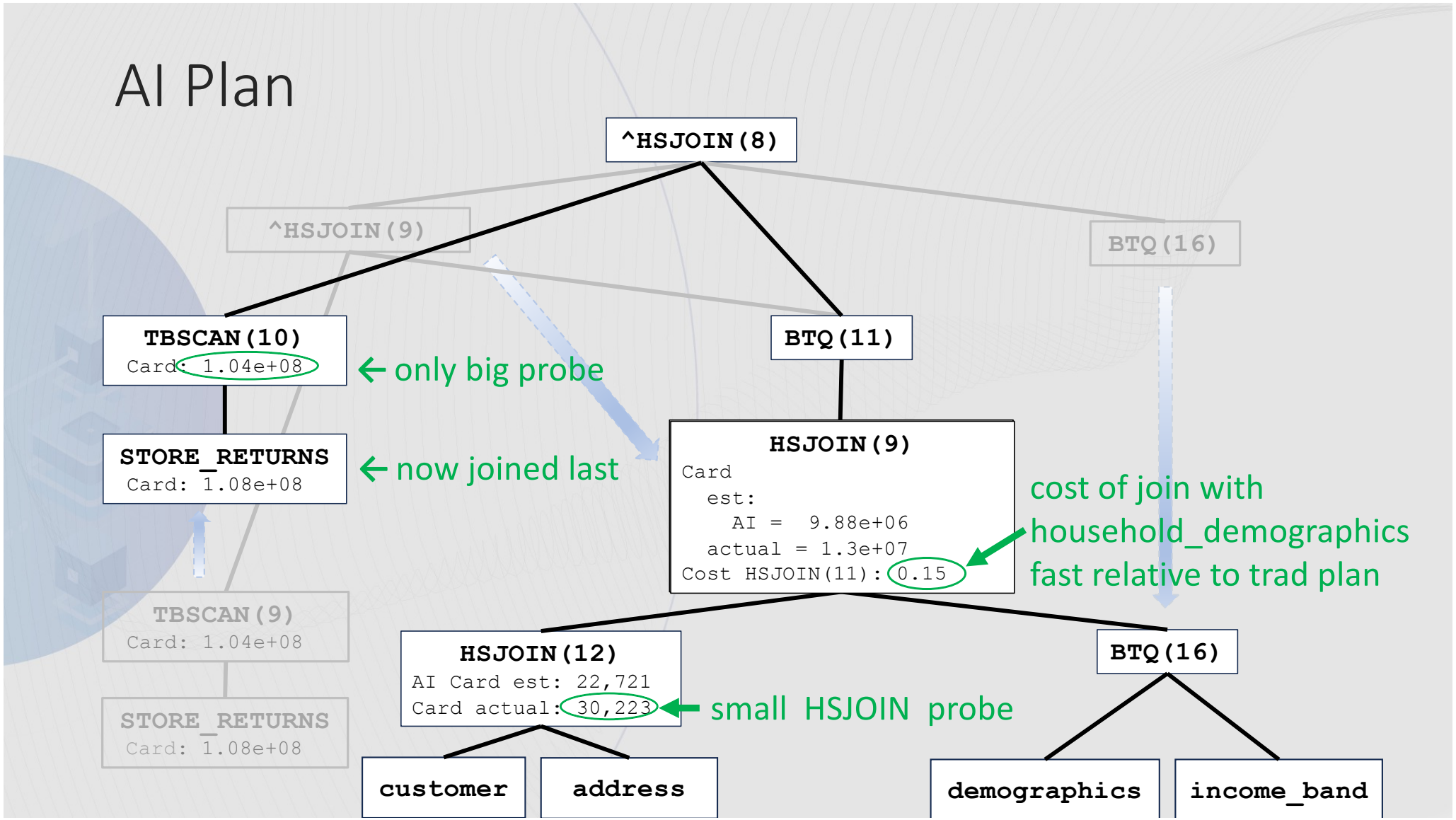
STORE_RETURNS

Card(est): 1.08e+08

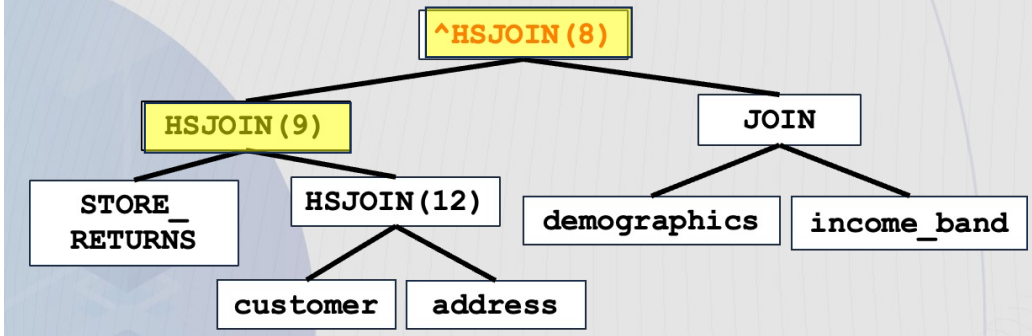
Traditional Plan

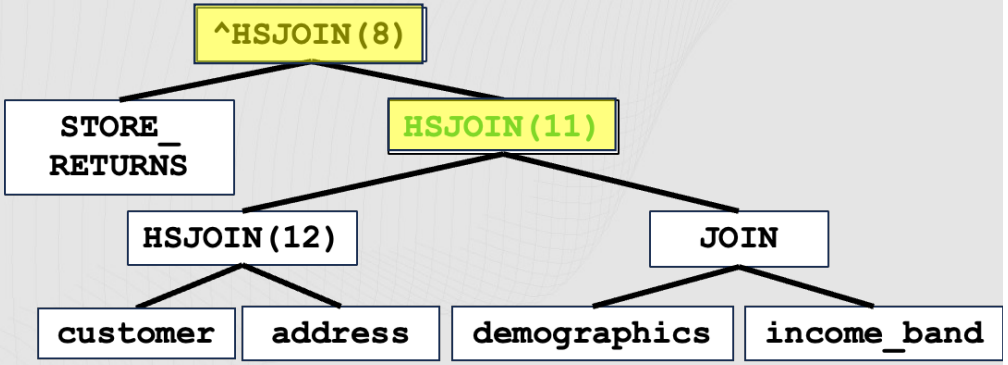


AI Plan

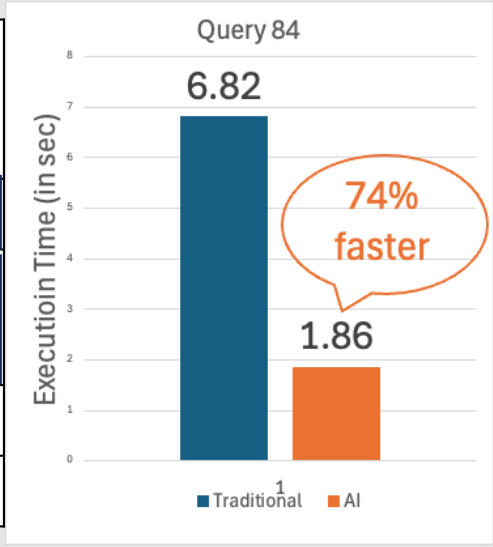

^HSJOIN(8)
^HSJOIN(9)
BTQ(16)
TBSCAN(10)
Card: 1.04e+08
only big probe
BTQ(11)
STORE_RETURNS
Card: 1.08e+08
now joined last
HSJOIN(9)
Card
est:
AI = 9.88e+06
actual = 1.3e+07
Cost HSJOIN(11): 0.15
cost of join with household_demographics fast relative to trad plan
TBSCAN(9)
Card: 1.04e+08
STORE_RETURNS
Card: 1.08e+08
HSJOIN(12)
AI Card est: 22,721
Card actual: 30,223
small HSJOIN probe
BTQ(16)
customer
address
demographics
income_band

Costs and Execution Times



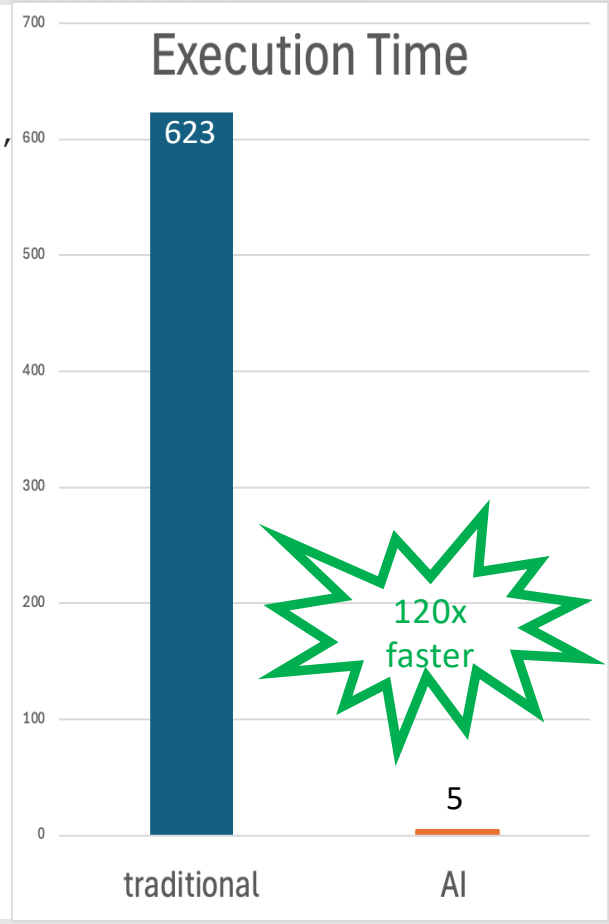


	Join in trad plan	Trad Plan Cost	Join in AI plan	AI Plan Cost
JOIN with store_returns	HSJOIN(9)	486.4	HSJOIN(8)	454.37
Join with household_demographics	HSJOIN(8)	49.9	HSJOIN(11)	0.15
Total join cost		536.3		454.52
TOTAL PLAN COST		36,362		36,280

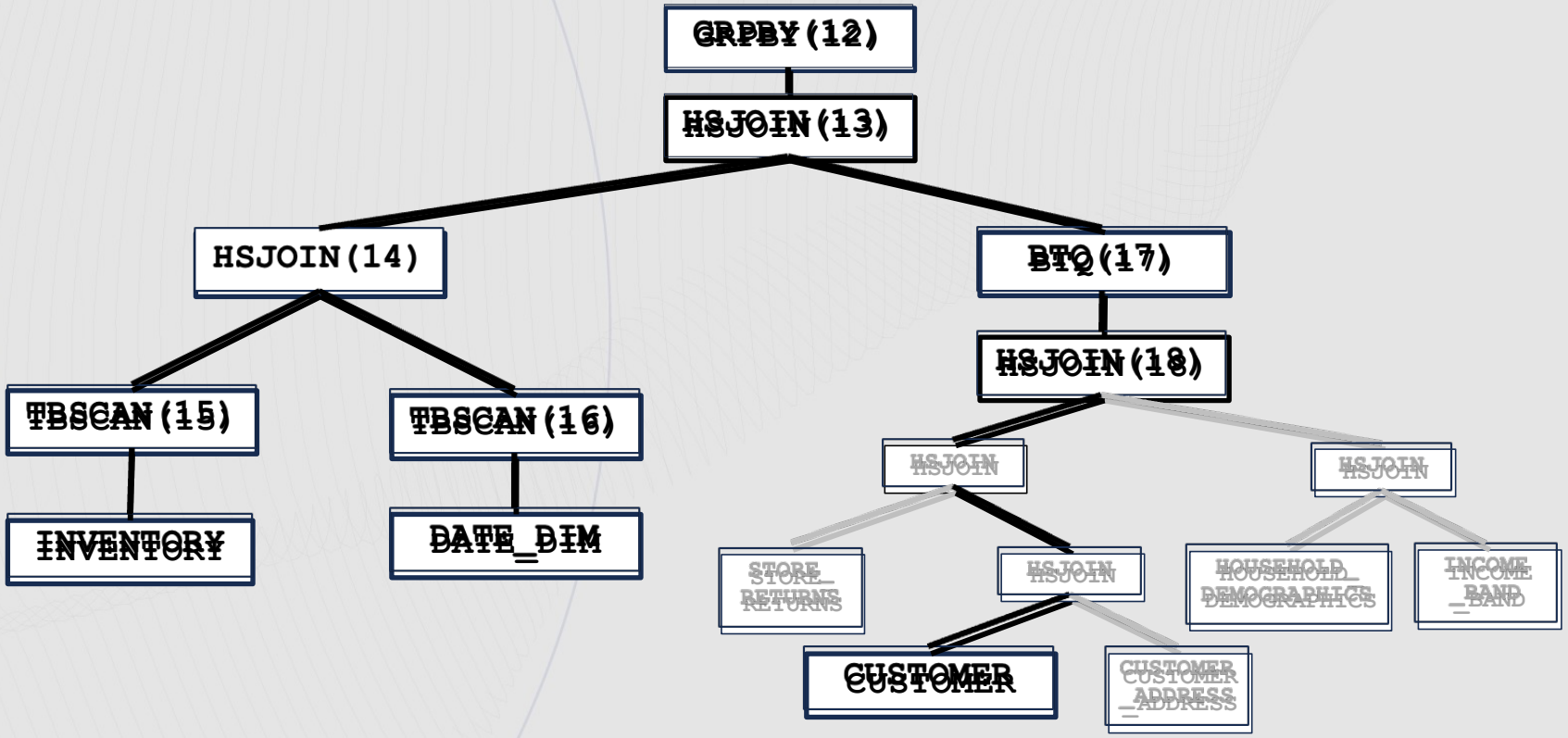


100 Times Faster

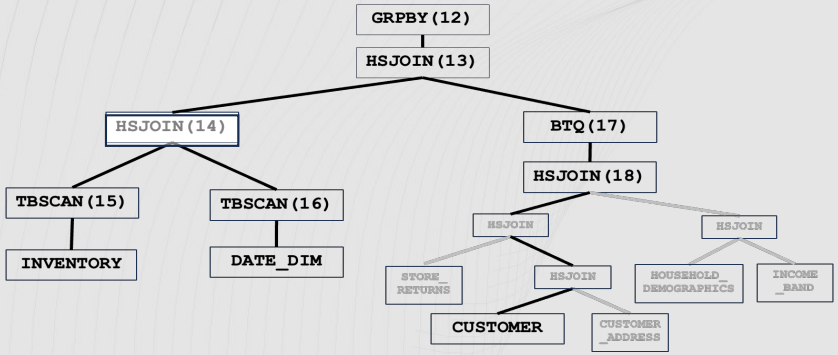
```
Select c_customer, c_last_name, c_first_name,
From customer, customer_address, customer_demographics,
     household_demographics, income_band_band, store_returns,
     inventory, date_dim, item
Where ca_city = 'Greenfield' and
      c_current_addr_sk = ca_address_sk and
      ib_lower_bound >= 11660 and
ib_upper_bound <= 11660 + 50000 and
      ib_income_band_sk = hd_income_band_sk and
      cd_demo_sk = c_current_cdemo_sk and
      hd_demo_sk = c_current_hdemo_sk and
      sr_cdemo_sk = cd_demo_sk and
      sr_item_sk = i_item_sk and
      inv_item_sk = i_item_sk and
      inv_date_sk = d_date_sk and
      d_month_seq between 1193 and 2000
group by c_customer_id, c_last_name, c_first_name
```



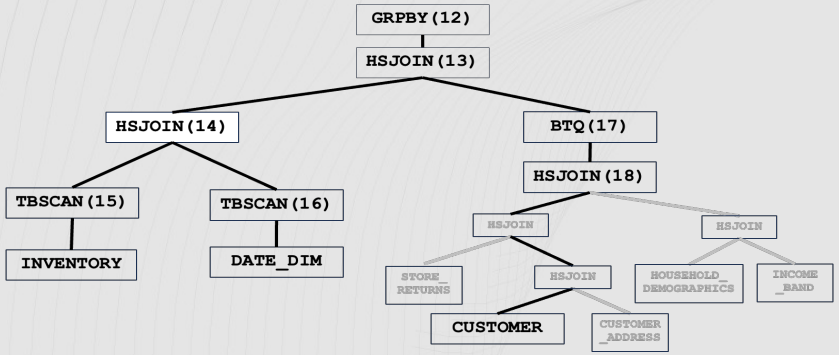
Traditional Plan



Traditional Plan



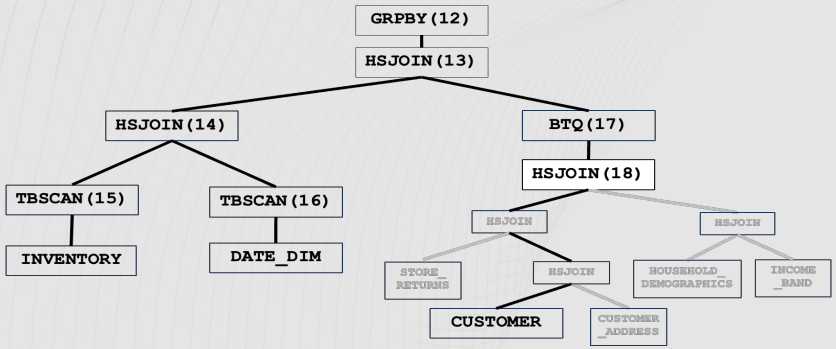
Traditional Plan



HSJOIN(14)

```
Card est: trad = 4.49e+07   ← 50% too low
          AI   = 1.16e+08   ← ~50% closer
Actual card:     9.25e+07
```

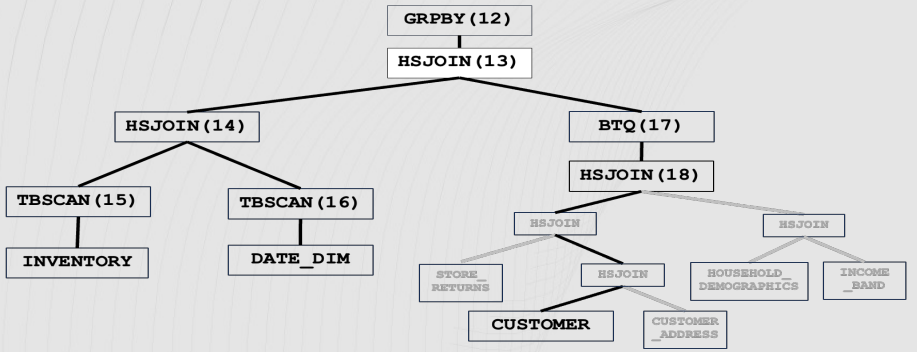
Traditional Plan



HSJOIN(18)

```
Card est:
   trad      =   59,493   ← 2% of actual
   AI        =  422,902   ← ~10x better
Card actual: 2.5e+06
```

Traditional Plan



HSJOIN(13)

```
Cardinality:
   estimate: traditional(trad)  = 1.09e+08
             AI                 = 9.76e+08
   actual                       = 5.19e+10
Cost:
  HSJOIN(13): trad =    517
              AI   =  2,491
  OPERATORS 13-18: trad = 72,651
                   AI   = 72,651
```

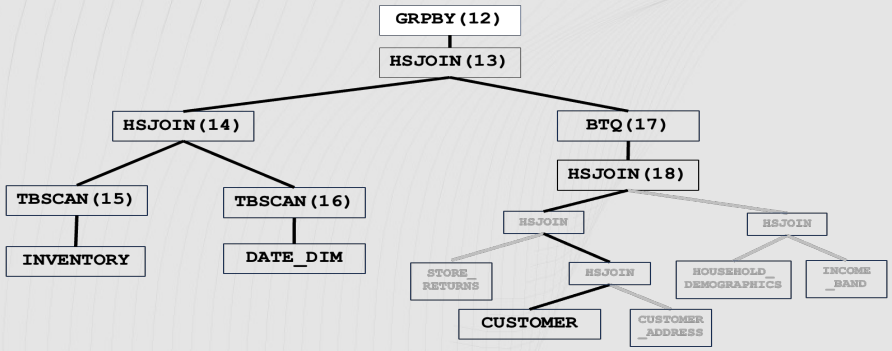
← 2% of actual

← ~10x better

← Cost under estimated

← 5x

Traditional Plan



GRPBY(12)

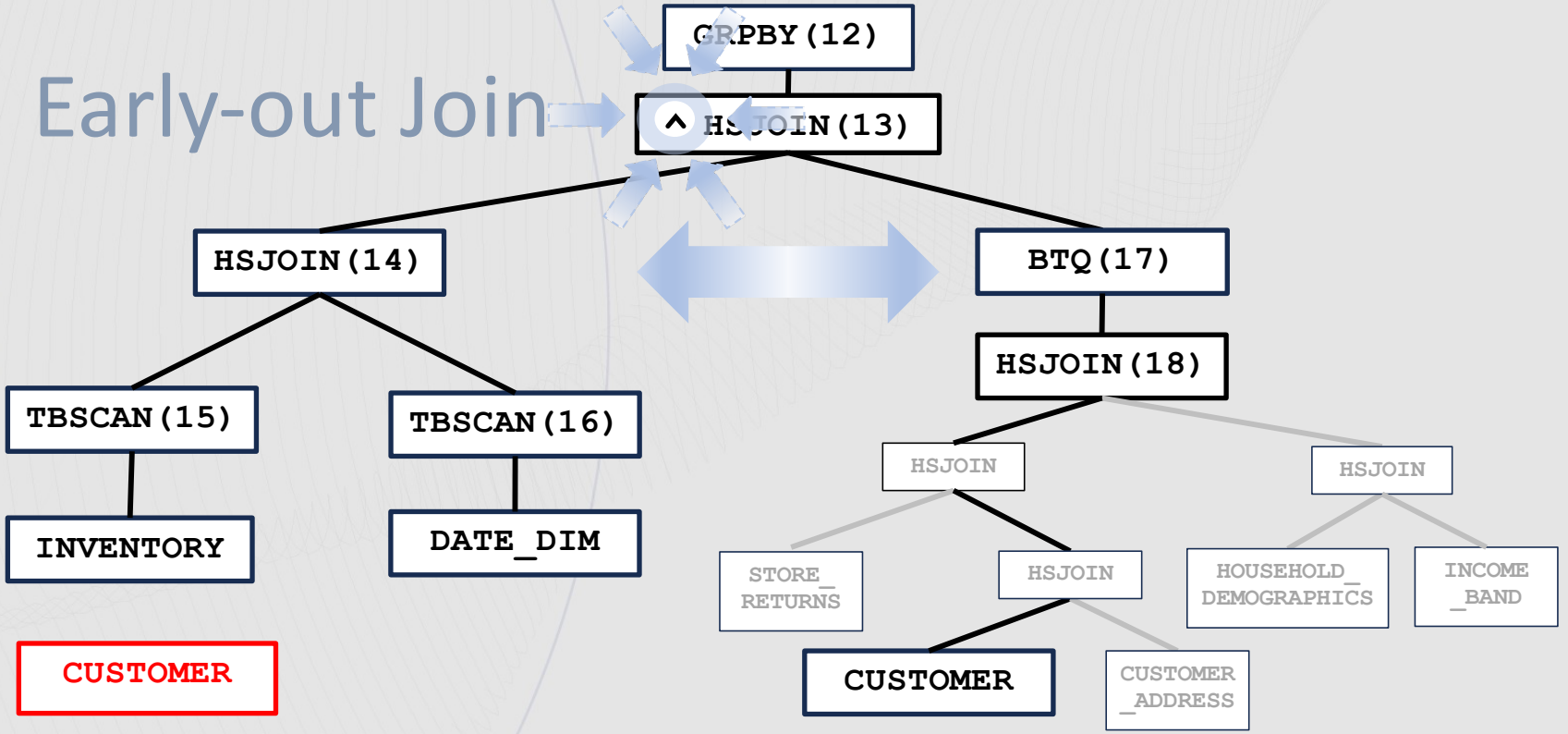
```
Cost:
   GRBPY(12): trad =     585
              AI   = 333,180
   OPERATORS 12-18: trad = 71,814
                    AI   = 405,831
```

← **500x** under costed

← costly group by

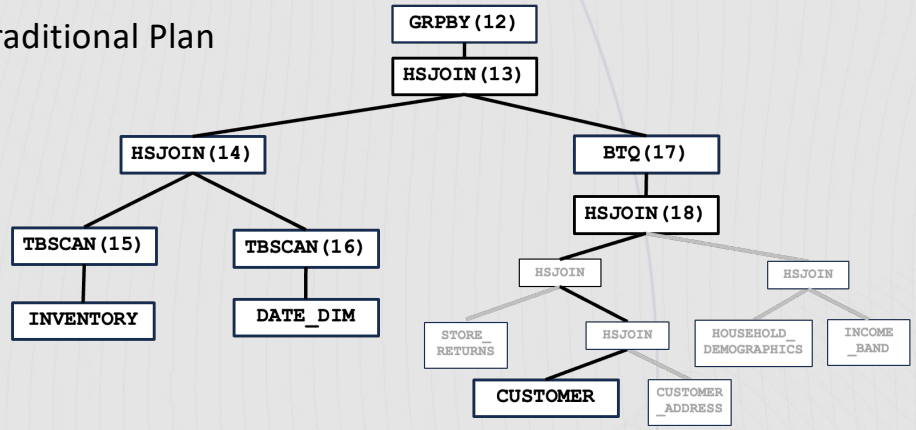
Traditional Plan

Early-out Join

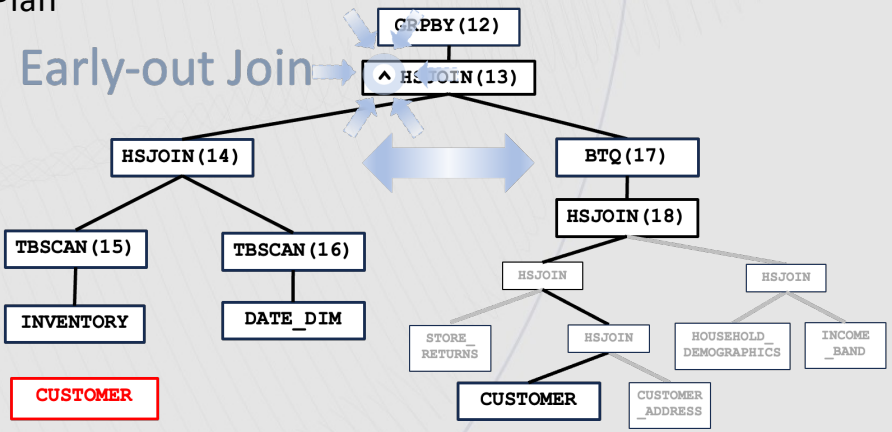


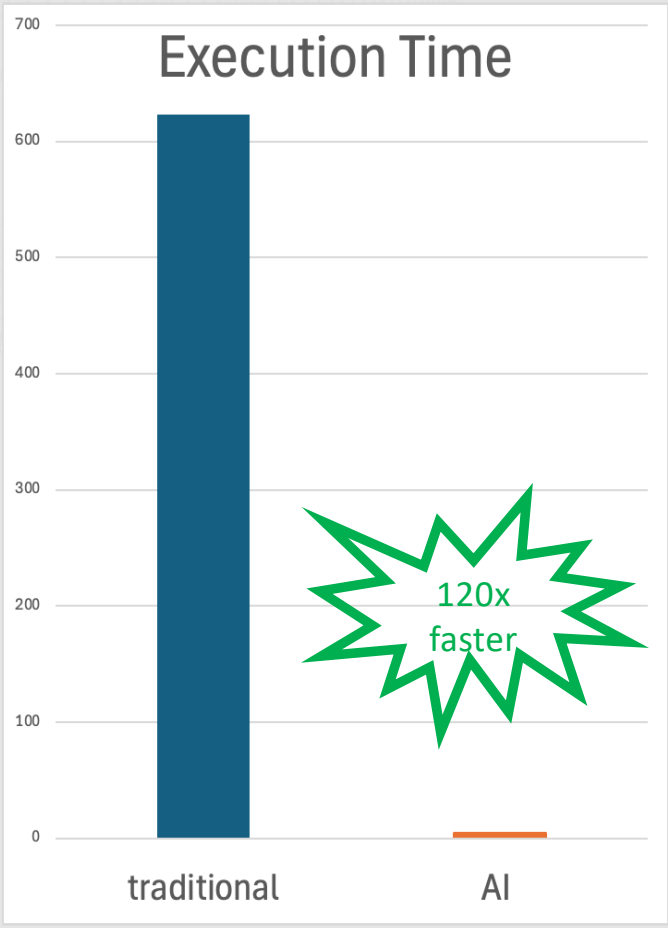
Better Plans → Faster Queries

Traditional Plan



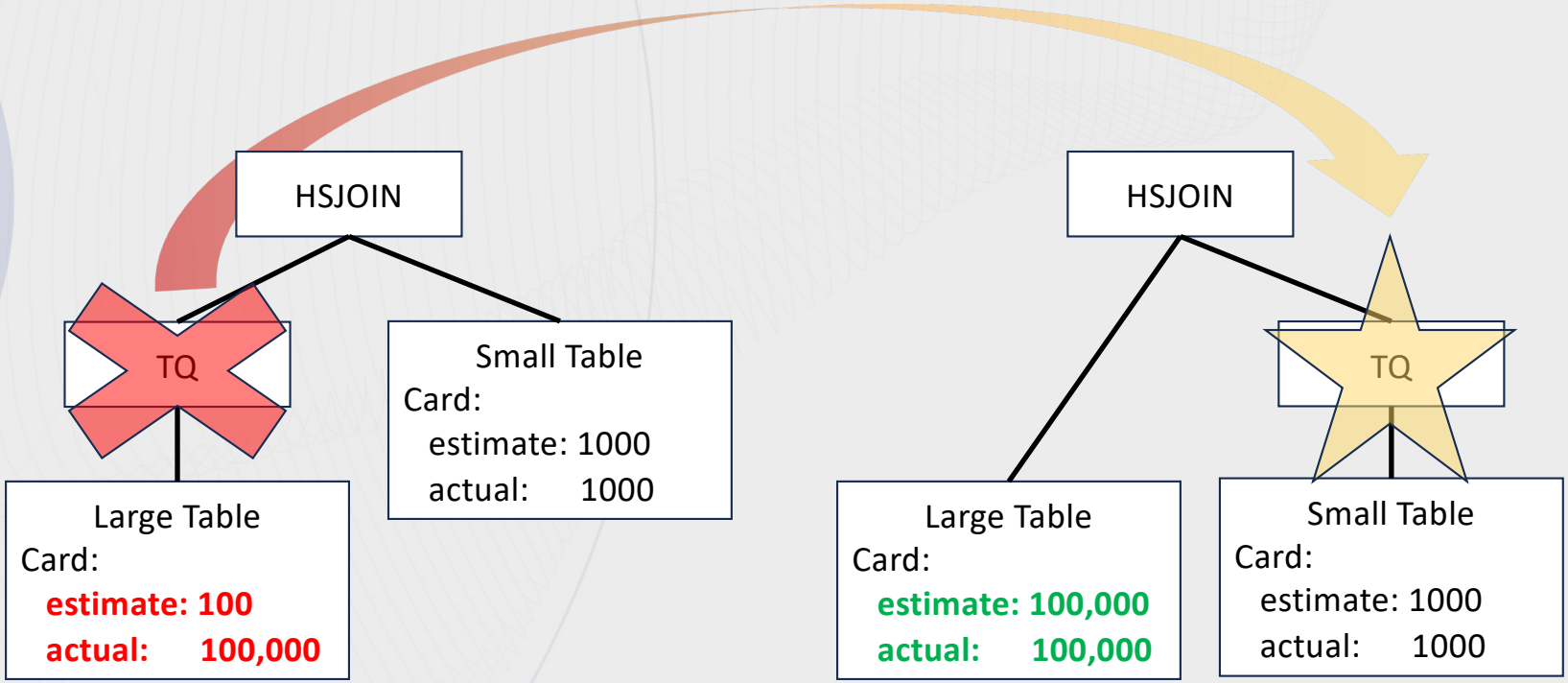
AI Plan



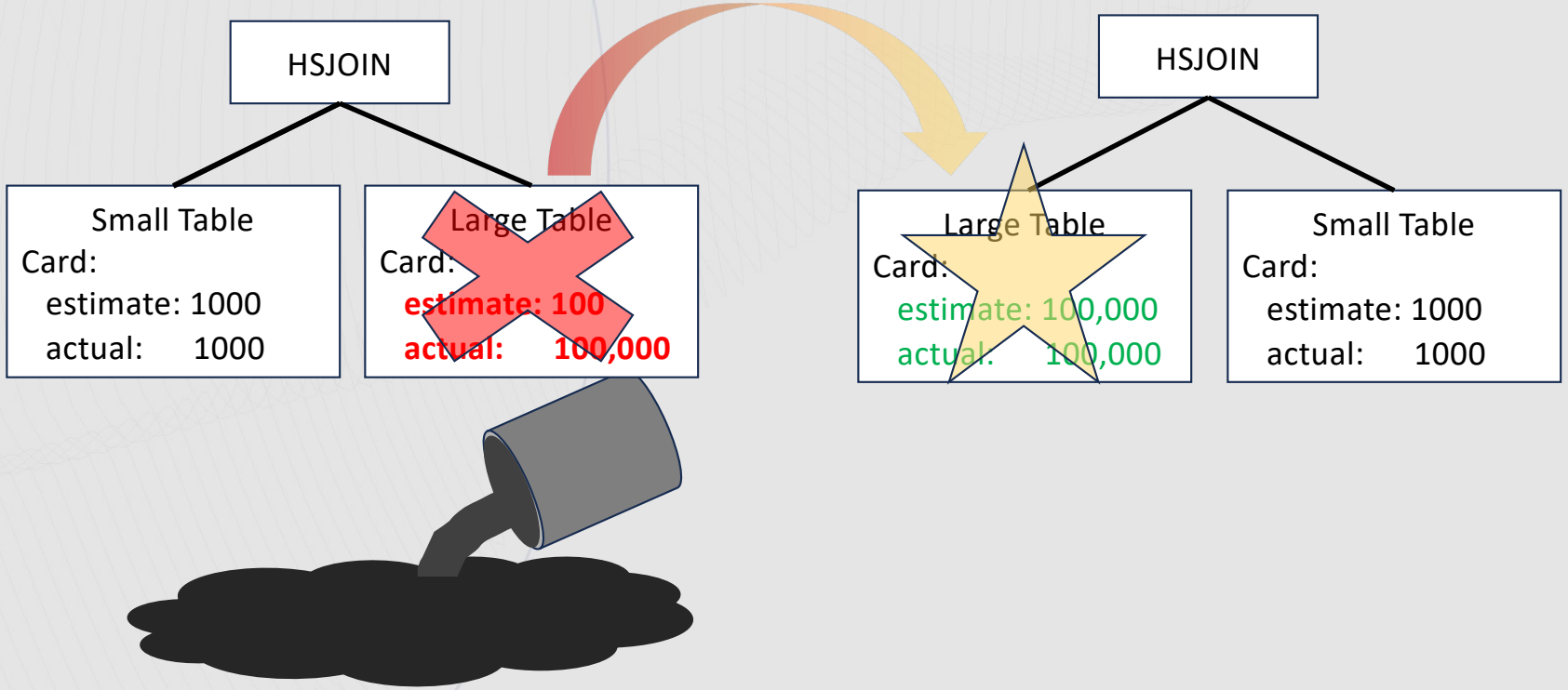


Other Ways AI Improves Performance:

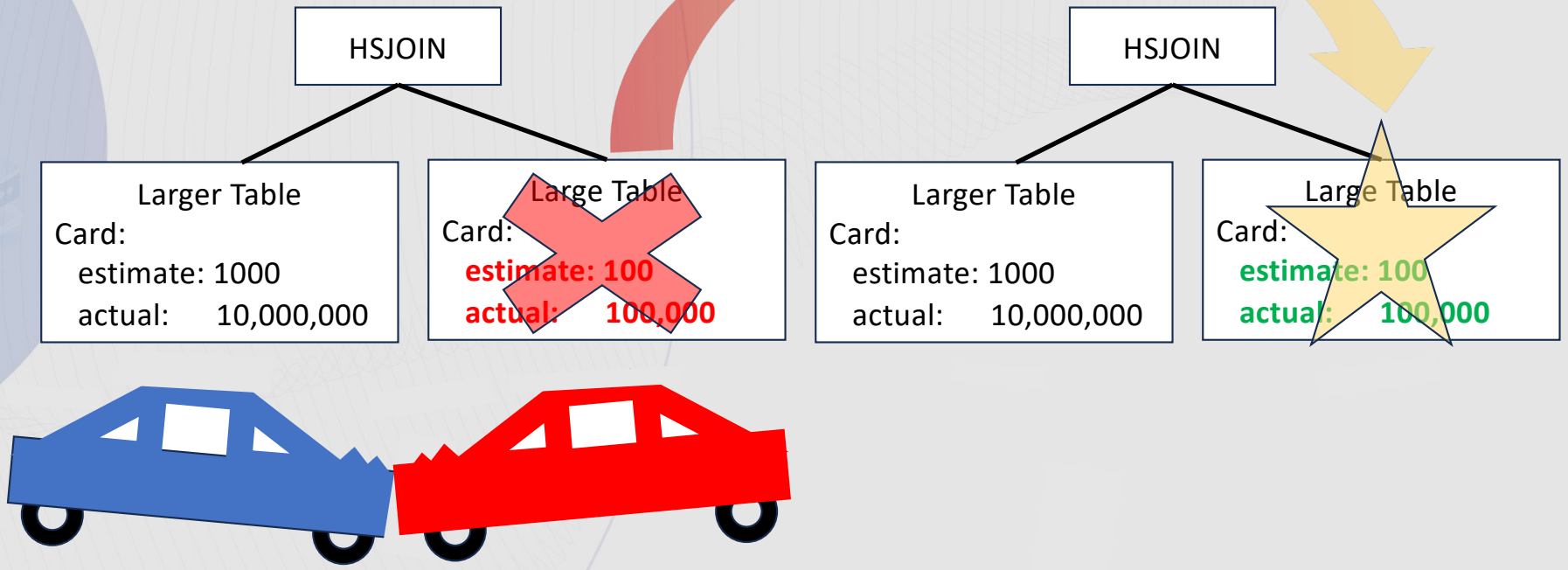
1) Avoids Repartitioning Unexpectedly Large Tables:



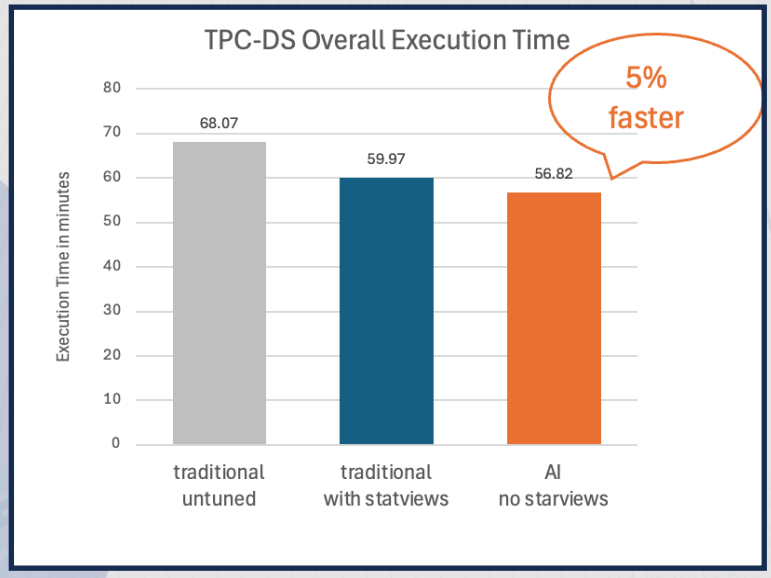
2) Prevents hash join spills

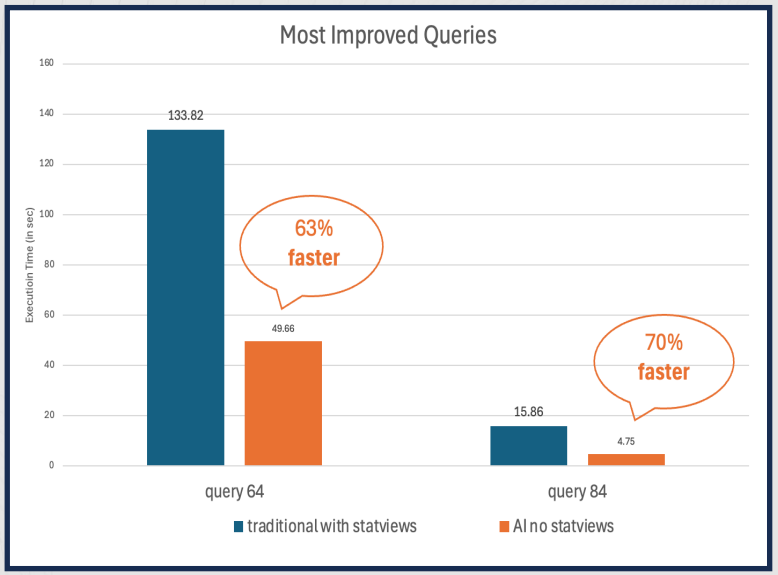


3) Reduces hash collisions:



AI vs Statistical Views





	Statview	AI
Effort to implement	40 hours	0
Runstats execution time	14 hours	3 minutes
TOTAL	54 hours	3 minutes

Model Training Time

GOAL: Minimize Resources

Goal:
Keep training time within 60s

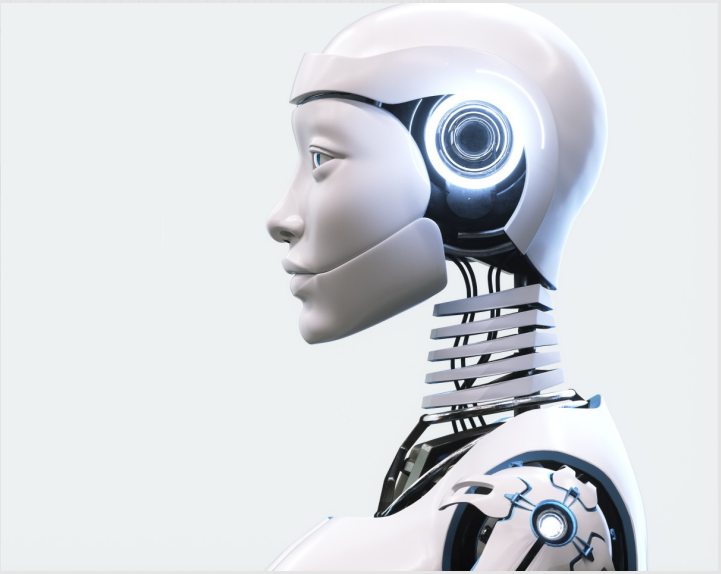
TABLENAME	TIME (S)
CALL_CENTER	0.01
WAREHOUSE	0.03
INCOME_BAND	0.04
REASON	10.47
HOUSEHOLD_DEMOGRAPHICS	16.54
WEB_PAGE	16.59
INVENTORY	16.47
STORE	14.86
SHIP_MODE	24.92
PROMOTION	21.33
CUSTOMER_DEMOGRAPHICS	22.56
TIME_DIM	24.99
WEB_SITE	28.28
CUSTOMER_ADDRESS	31.87
STORE_RETURNS	25.05
WEB_RETURNS	28.75
CATALOG_RETURNS	30.52
STORE_SALES	36.96
CATALOG_SALES	45.01
WEB_SALES	48.61
CATALOG_PAGE	46.49
DATE_DIM	41.71
CUSTOMER	44.66
ITEM	53.33

MAX	53.33(s)
AVERAGE	26.25(s)
TOTAL	10.5(m)

Models tiny but mighty

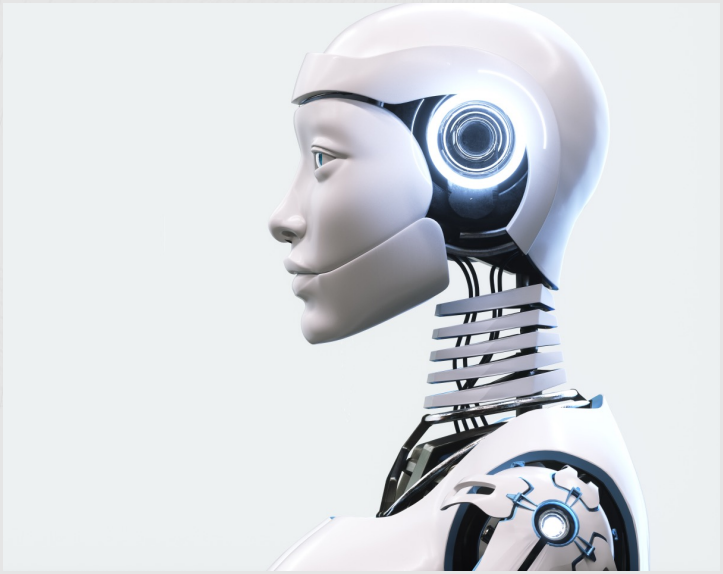
Summary

- Reliable
- Simplify
- Minimize Resources



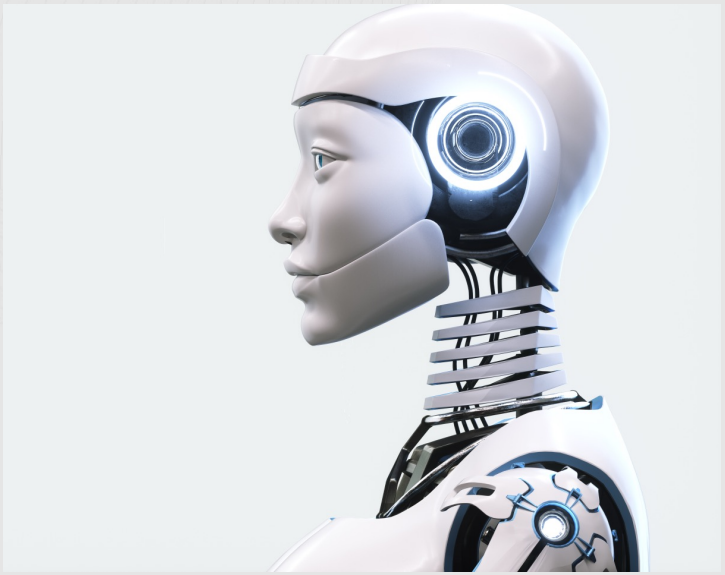
Summary

Reliable	Improved cardinality, plans and query performance
Simplify	
Minimize Resources	



Summary

Reliable	Improved cardinality, plans and query performance
Simplify	Automation reduces need for tuning
Minimize Resources	



Summary

Reliable

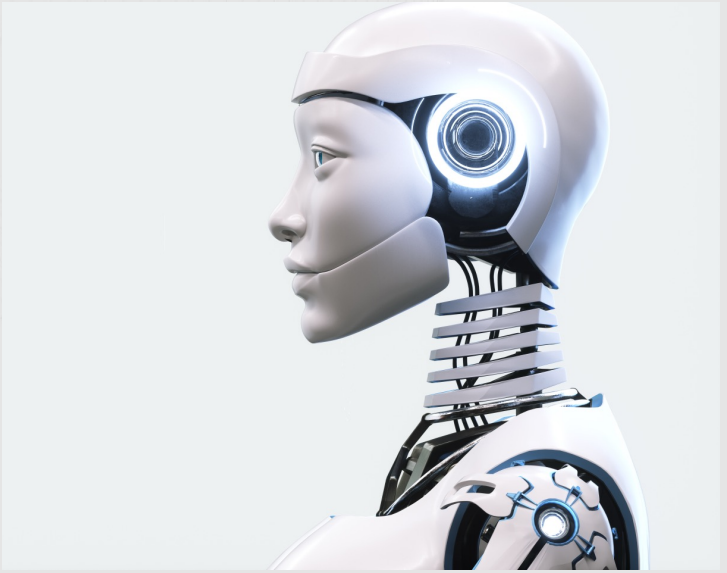
Improved cardinality, plans and query performance

Simplify

Automation reduces need for tuning

Minimize Resources

- hardware
- Time to derive insight
- Expertise
- No new models required
- Models are tiny



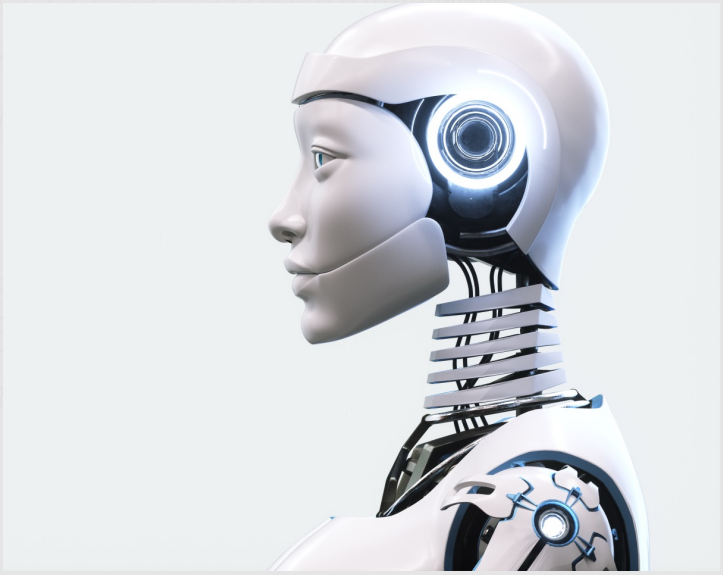
Summary

Reliable — Improved cardinality, plans and query performance

Simplify — Automation reduces need for tuning

Minimize Resources

- hardware
- Time to derive insight
- Expertise
- No new models required
- Models are tiny

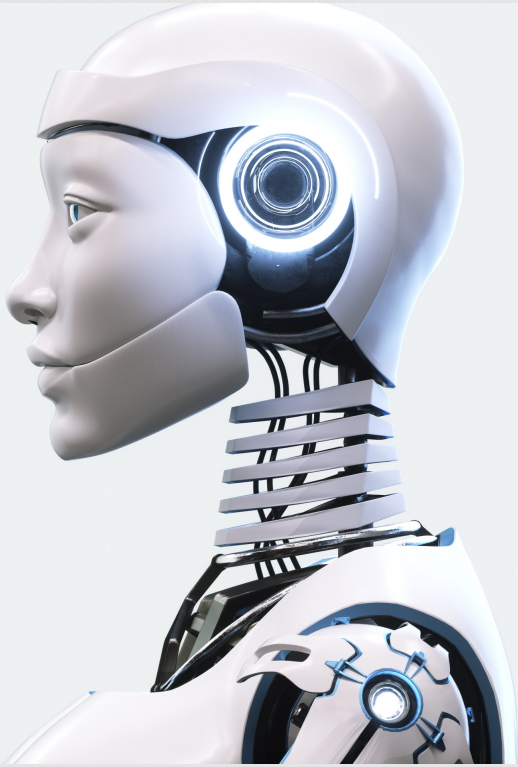


COST SAVINGS!

The journey continues…

Infuse

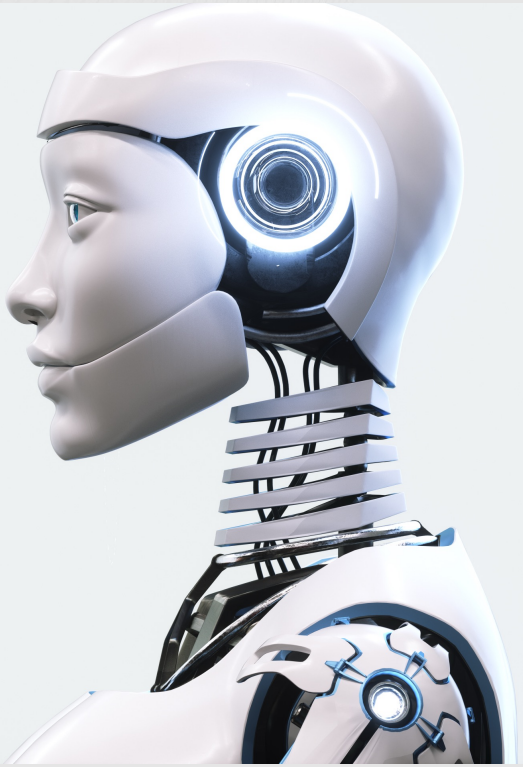
- Single table cardinality estimation since 12.1 GA
- Join cardinality estimation in 12.1.3
- Next steps:
  - Join card string support
  - Multi-dimension joins
    - Will reduce the need for column group stats
  - and …



Coming Soon

Infuse

AI Query Rewrite



Learn More

- Documentation: Db2 Version 12.1 - AI Query Optimizer
- Webinar: The Db2 AI Query Optimizer – Simplifying Query Performance Tuning
- Webinar: Experience the Power and Simplicity of the AI Query Optimizer in Db2 12.1.3
- Blog: Significant Performance Improvements with AI Join Cardinality Estimates in Db2 version 12.1.3
- Video: IBM Db2 AI Query Optimizer
- Video: Ai Query Optimizer in DB2 12.1.3 - technical insights of JOIN queries infusing AI

Call for Sessions

Submission deadline: May 22

IBM TechXchange 2026

October 26–29 | Atlanta, GA

Clients, Partners & IBMers are eligible to submit sessions

Share your expertise with thousands of technical professionals!

What we are looking for

- Real-life use cases and implementation stories
- Technical deep dives (non-promotional)
- Demonstrated expertise with IBM products
- Sessions aligned to 2026 priorities: enterprise AI applications, key products, technical skills
- A compelling title and concise, clear abstract

Session types

Technology Breakout (45 minutes)
In-depth technical session featuring architectures, demos, best practices, client/partner stories

Tech Talk (20 minutes)
Fast-paced spotlight on use cases, technical trends, and special-topic insights.

Client & partner submissions only for Tech Talks.

Technology areas

AI Applications | Data Management | Cloud & Platform Engineering | Infrastructure | Security, Risk & Compliance

Special perks

- A complimentary full conference pass (that's a $1599 USD value!)
- Speaker training and content support
- Social tools to promote your session

Submission deadline: May 22

Get started →