IDUG

Db2 SQL Performance for Application Developers

David Morris, Leidos

Session Code: E03

Boston, MA

Database Performance Aspects

Application

Database
Structures

Db2 Server

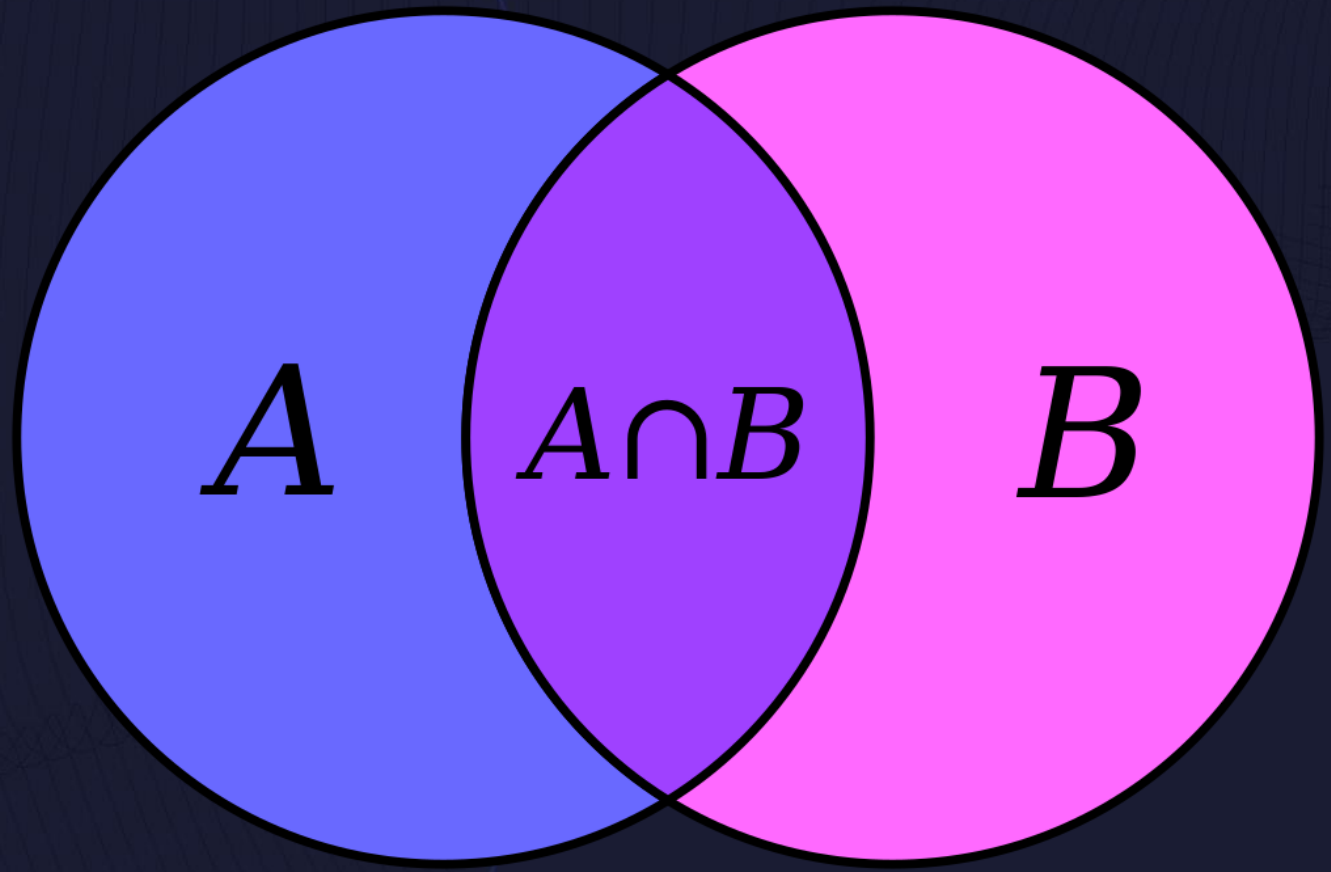
Database Performance Indicators

CPU

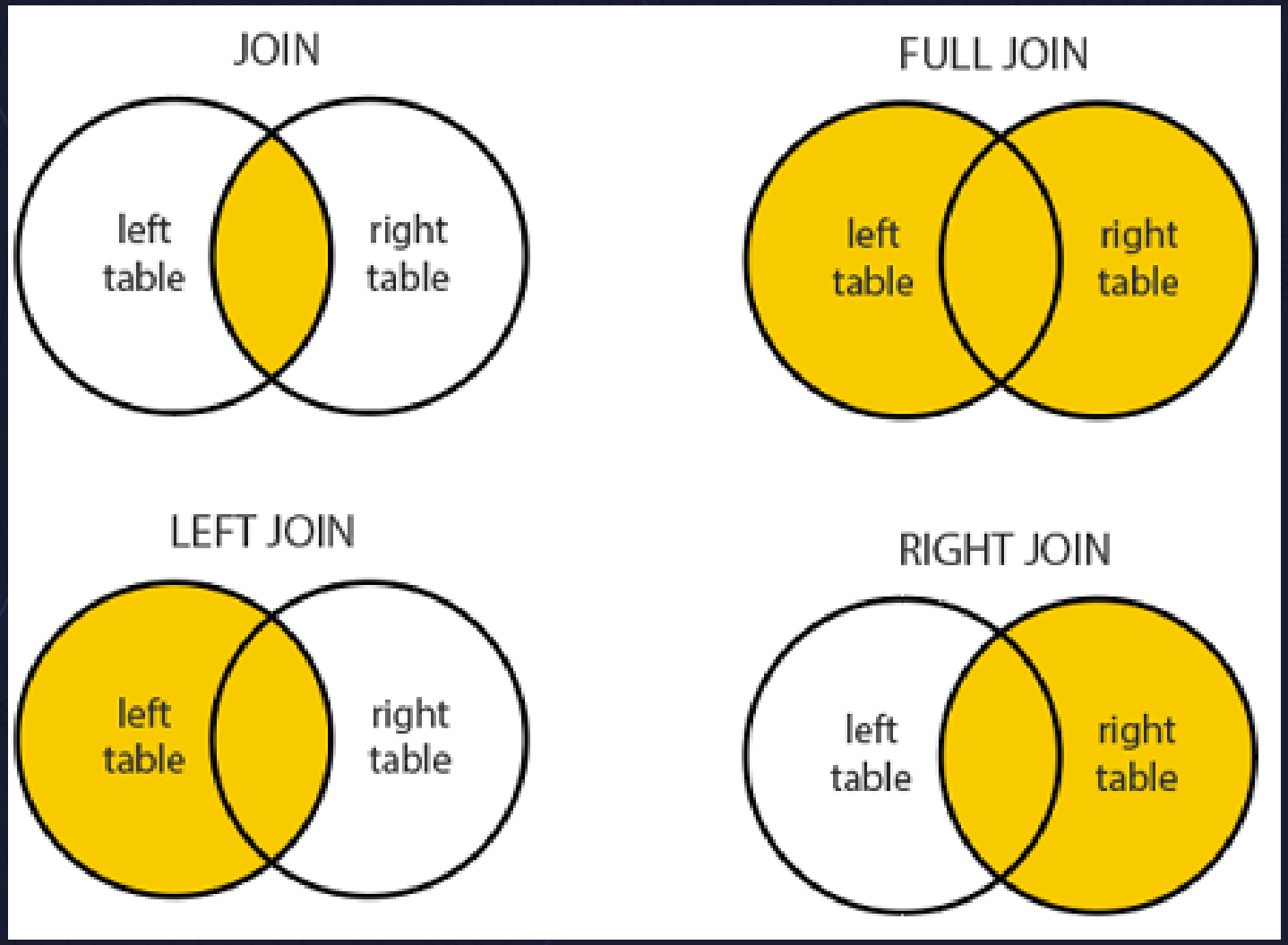
I/O

Concurrency

Relation Databases – Set Theory

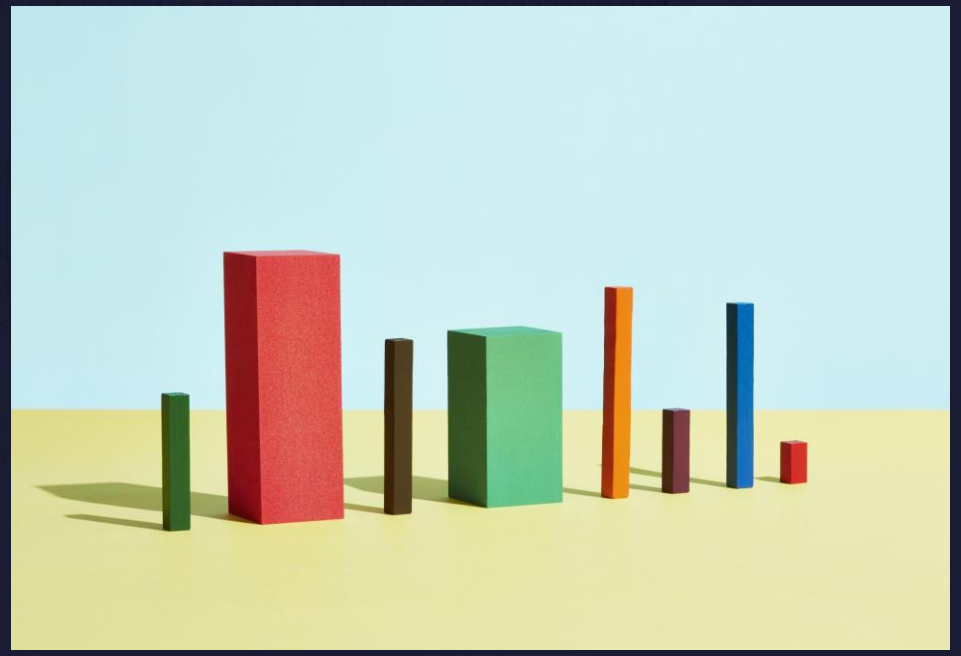


Joins


JOIN
left table
right table
FULL JOIN
left table
right table
LEFT JOIN
left table
right table
RIGHT JOIN
left table
right table

Sorting

- ORDERED BY
- DISTINCT
- Limit columns to sort
- Favor NOT EXISTS over NOT Indexes
- GROUP BY
- GROUP BY and ORDERED BY



Predicates

- Stage 1 Predicate
  - SELECT LASTNAME, FIRSTNAME FROM STUDENT WHERE STUDENT_ID = 21
- Stage 2 Predicate
  - SELECT LASTNAME, FIRSTNAME FROM STUDENT WHERE STUDENT_ID BETWEEN 100 AND 300

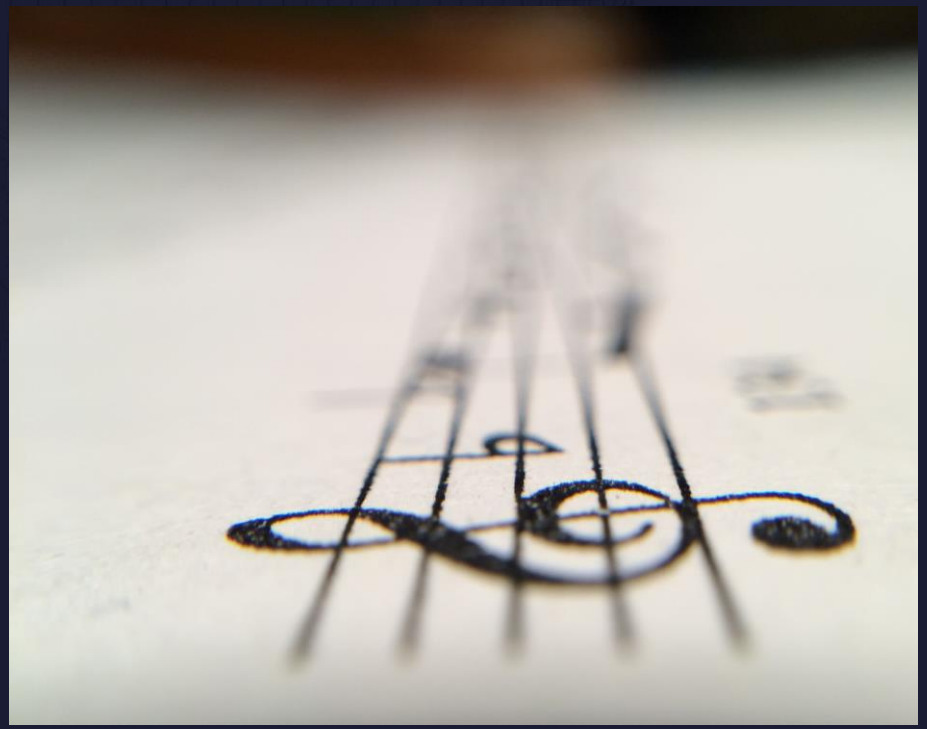
Db2 Functions

- Common Db2 Functions
  - SUM
  - DATE
  - REPLACE
  - AVG
  - YEAR
- Functions are Stage 2 Predicates
  - Can be rewritten as Stage 1 Predicates

```
SELECT FIRST_NAME, LAST_NAME FROM STUDENT
 WHERE SUBSTR(LAST_NAME,1,1)='A'
```

TO

```
SELECT FIRST_NAME, LAST_NAME FROM STUDENT
 WHERE LAST_NAME LIKE 'A%'
```



SQL Isolation Levels

- CS - Cursor stability
  - Only committed data can be read
- RR - Repeatable Read
  - useful for programs that require consistency in rows that may be accessed twice in one execution of the application.
- RS - Read stability
  - A retrieved row or page is locked until the end of the unit of work.
- UR - Uncommitted Read
  - Program that can read data that has changed but not yet committed.

Indexes

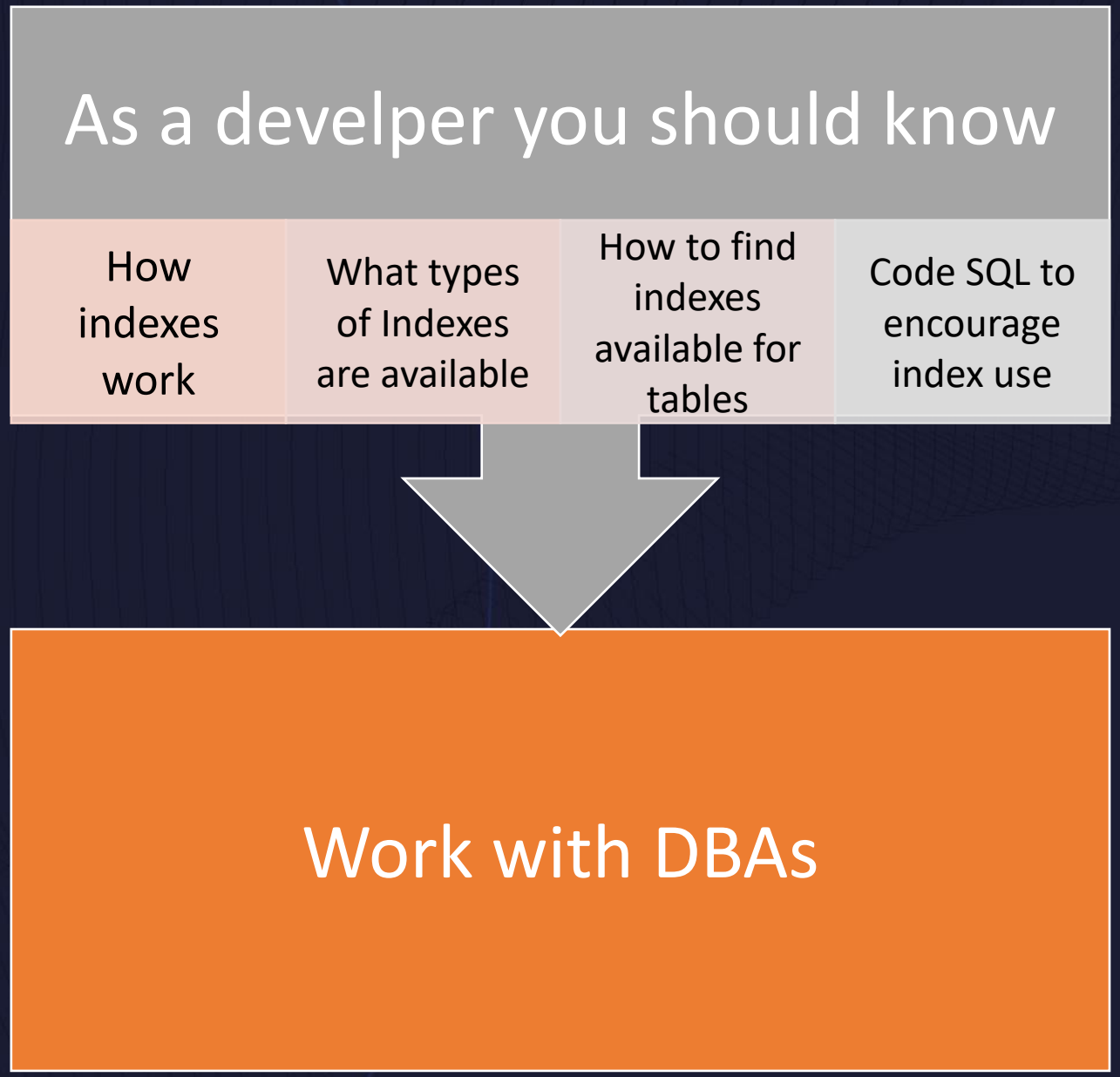

As a develper you should know
How indexes work
What types of Indexes are available
How to find indexes available for tables
Code SQL to encourage index use
Work with DBAs

Table Scan

DB2 CANNOT RESOLVE A QUERY

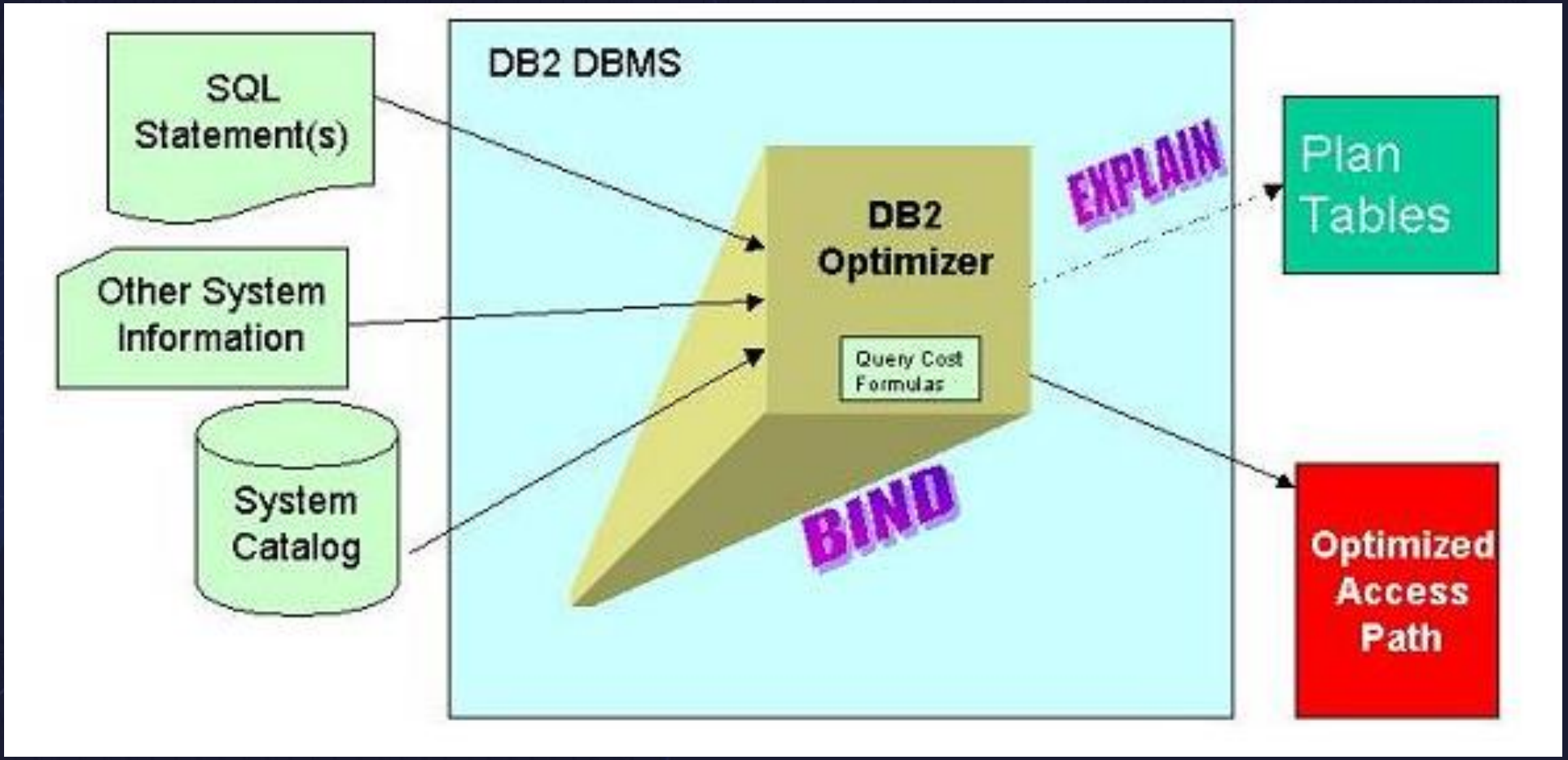
SCANS ARE EXPENSIVE

TABLE SCANS MAY BE BEST WAY TO RESOLVE A QUERY

Cursors

- DECLARE cursor_name CURSOR IS SELECT ... FROM ...

Db2 Optimizer



SQL Statement

- Do not use SELECT *
- Select specific fields required only
  - SELECT FIRST_NAME, LAST_NAME FROM STUDENT
- Do not ask what you already know
  - SELECT STUDENT_ID, FIRST_NAME, LAST_NAME FROM STUDENT WHERE STUDENT.STUDENT_ID = 887527;

Multiple ways to code a SQL statement

- Formulate multiple ways of writing SQL to improve performance
  - Examples:
    - SELECT COURSE_NAME FROM COURSES WHERE COURSE_NAME LIKE 'A%' OR COURSE_NAME LIKE 'J%';
    - SELECT COURSE_NAME FROM COURSES WHERE COURSE_NAME IN ('Angular', 'JavaScript');
    - SELECT COURSE_NAME FROM COURSES WHERE COURSE_NAME='Angular' OR COURSE_NAME='JavaScript';

Multiple ways to code a SQL statement

- Use Stage 1 Predicates over Stage 2 Predicates
  - Stage 2
    - SELECT LASTNAME, FIRSTNAME FROM STUDENT WHERE STUDENT_ID BETWEEN 100 AND 300
  - Stage 1
    - SELECT LASTNAME, FIRSTNAME FROM STUDENT WHERE STUDENT_ID = 150

Multiple ways to code a SQL statement

- Rewrite SQL to achieve Indexability
  - SELECT FIRST_NAME, LAST_NAME FROM STUDENT WHERE LAST_NAME BETWEEN 'A' AND ‘G’

TO

SELECT FIRST_NAME, LAST_NAME FROM STUDENT WHERE LAST_NAME NOT BETWEEN 'H' AND 'Z'

Multiple ways to code an SQL statement

- Favor JOINS instead of Subqueries
  - SELECT COUNT(COURSE_ID) FROM CLASSES WHERE CLASSES.COURSE_ID IN (SELECT COURSE_ID FROM COURSES WHERE COURSE_NAME LIKE '%Db2%')

    TO

  - SELECT COUNT(CLASSES.COURSE_ID) FROM CLASSES, COURSES WHERE CLASSES.COURSE_ID = COURSES.COURSE_ID AND COURSES.COURSE_NAME LIKE '%Db2%';

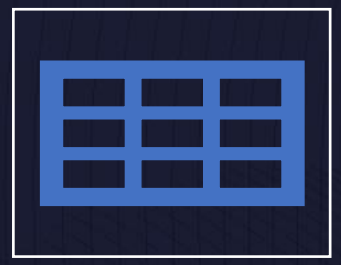
Multiple ways to code an SQL statement

- Filtering in OUTER JOINS
  - SELECT CLASS_ID, LOCATION, DURATION, FREQUENCY FROM CLASSES LEFT JOIN COURSES ON CLASSES.COURSE_ID = COURSES.COURSE_ID **WHERE** COURSES.COURSE_NAME = 'Db2 for Developers'
  - SELECT CLASS_ID, LOCATION, DURATION, FREQUENCY FROM CLASSES LEFT JOIN COURSES ON CLASSES.COURSE_ID = COURSES.COURSE_ID **AND** COURSES.COURSE_NAME = 'Db2 for Developers';

SQL Tweaks

FETCH FIRST

Example: FETCH FIRST 10 ROWS ONLY;



Check for Existence in a Table

Example: FETCH FIRST 1 ROW ONLY;

SQL Tweaks

- CASE Expressions

```
SELECT FIRST_NAME, LAST_NAME,
CASE
  WHEN STATE IN ('AZ', 'NM', 'CA', 'NV', 'UT', 'CO', 'TX', 'OK') THEN 'REGION 1’
  WHEN STATE IN ('MD', 'VA', 'DC', 'DE', 'WV', 'PA', 'NJ') THEN 'REGION 2’
  WHEN STATE IN ('NY', 'CT', 'VT', 'MA', 'NH', 'ME', 'RI') THEN 'REGION 3’
  ELSE 'REGION 4'
END
FROM STUDENT;
```

Advanced SQL

- Avoid Cartesian Products
- Code LEFT OUTER JOIN instead of RIGHT OUTER JOIN
- Use appropriate Data/Time data type
  - Built in date and time arithmetic

Demo

DB2 v11.5.7000.1973

Processor Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)

Installed Physical Memory (RAM) 8.00 GB

OS Name Microsoft Windows 11 Home

250 GB SSD

Angular 13.2

Node v16.13.1

Demo

DB2 v11.5.7000.1973

Processor Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)

Installed Physical Memory (RAM) 8.00 GB

OS Name Microsoft Windows 11 Home

250 GB SSD

Angular 13.2

Node v16.13.1

- DEMO UNIVERSITY
  - STUDENT 1,000,000
  - COURSES 50
  - CLASSES 200,000
  - GRADES 4,000,000 ‘G’

Db2 SQL Performance Recap

- ORDERED BY
- Minimize passes thru the data
- Prefer SQL logic vs. Application logic
- Write application unit tests
- Avoid ORM SQL generators
- CODE commits in your programs

- Use Db2 Utilities
- Load Utility for Bulk Inserts
- Use SQL Functions
- Prefer Stage 1 Predicates vs. Stage 2
- Work with DBA
- Explains plan
- Try different SQL queries
- DO NOT USE SELECT *

Thank You

Speaker: David Morris

Company: Leidos

Email Address: david.p.morris.jr@leidos.com

Session Code: E03

Please fill out your session evaluation before leaving!