Databases
Lab

SQL Exercises

Flavio Bertini

flavio.bertini@smartdata.cs.unibo.it

Different approaches, different styles

- If your are more “theoretic guys”, think first at relational algebra expressions, and then transform them to SQL queries.
- If you are more “geeks”, learning SQL will help you to write down relational algebra expressions.

How to solve the exercises?

Step 0

Even before reading the text of the exercise we have to ask ourselves:

0.1. Which **database** are we gonna use?

- Understand the database schema, in order to recognize entities and relations.

0.2. Which are the **primary keys** and the **foreign keys**?

- *Primary keys* are usually underlined, and *foreign keys* uniquely identify a row of another table.

Step 0.1: The database schema

Given the following schema:

LECTURER(<u>Id</u>, Surname, Dept)

STUDENT(<u>Id</u>, Surname)

COURSE(<u>Code</u>, Name)

EDITION(<u>Course</u>, <u>Year</u>, Lecturer)

EXAM(<u>Student</u>, <u>Course</u>, Year)

Which information can we derive?

Step 0.2: Primary and foreign keys

```
LECTURER(Id, Surname, Dept)
STUDENT(Id, Surname)
COURSE(Code, Name)
```

- The underlined attributes are *primary keys*.

```
EDITION(Course, Year, Lecturer)
EXAM(Student, Course, Year)
```

- A code identifies a course, but each course could have more than one edition: `Course` in `EDITION` is a *foreign key.*
- What about EXAM?

How to solve the exercises?

Step 1

Then, we can read the query, and we have to ask ourselves the following questions:

1.1. Which **relations** are involved?

- To list them in the **`FROM`** clause.

1.2. What should be the **resulting attributes** list?

- To list them in the **`SELECT`** clause.

1.3. Do we need to **filter the data**?

- *Left/Right/Outer join* in **`FROM`** clause.
- *Theta-join* or value selection in **`WHERE`** clause.
- Aggregated values in **`GROUPBY`** (**`HAVING`**) clause.

Summary of Syntax

Execution Order

```
5    SELECT Attributes + Expressions
1    FROM Tables + Joins
2 [  WHERE Conditions                ]
3 [  GROUP BY Attributes             ]
4 [  HAVING Aggregated conditions    ]
6 [  ORDER BY Ordering attribute     ]
```

Exercise 1

Given the following schema:

```
LECTURER(Id, Surname, Dept)
STUDENT(Id, Surname)
COURSE(Code, Name)
EDITION(Course, Year, Lecturer)
EXAM(Student, Course, Year)
```

Write a SQL query that returns the distinct surnames that belong both to students and to lecturers.

Exercise 1 - step-by-step

Step 1.1 - Let's start by identifying the involved relations, that is `LECTURER` and `STUDENT`.

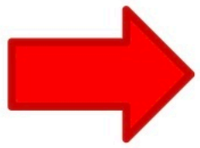
Step 1.2 - Then, to define the resulting attributes list, we have to use the `DISTINCT` keyword to avoid redundancies (i.e., duplicate `Surname`).

Step 1.3 - Finally, to filter the data we have two options:

1. use the intersection operator (too easy!)
2. perform a join between the two relations.

Exercise 1 - easy solution

```
SELECT DISTINCT Surname
FROM LECTURER
```



```
INTERSECT
SELECT DISTINCT Surname
FROM STUDENT
```

Exercise 1 - tricky solution

```
SELECT DISTINCT LECTURER.Surname
FROM LECTURER, STUDENT
WHERE LECTURER.Surname =
      STUDENT.Surname
```

implicit equi-join

Exercise 1 - good solution

```
SELECT DISTINCT L.Surname
FROM LECTURER AS L JOIN
     STUDENT AS S ON
     L.Surname = S.Surname
```

explicit equi-join

Word of Wisdom

There's no such thing as:

~~*There is only one correct solution!*~~

Each of us has a different "style" by following different pragmatisms, as in programming languages.

The important thing is to leave no room for bugs.

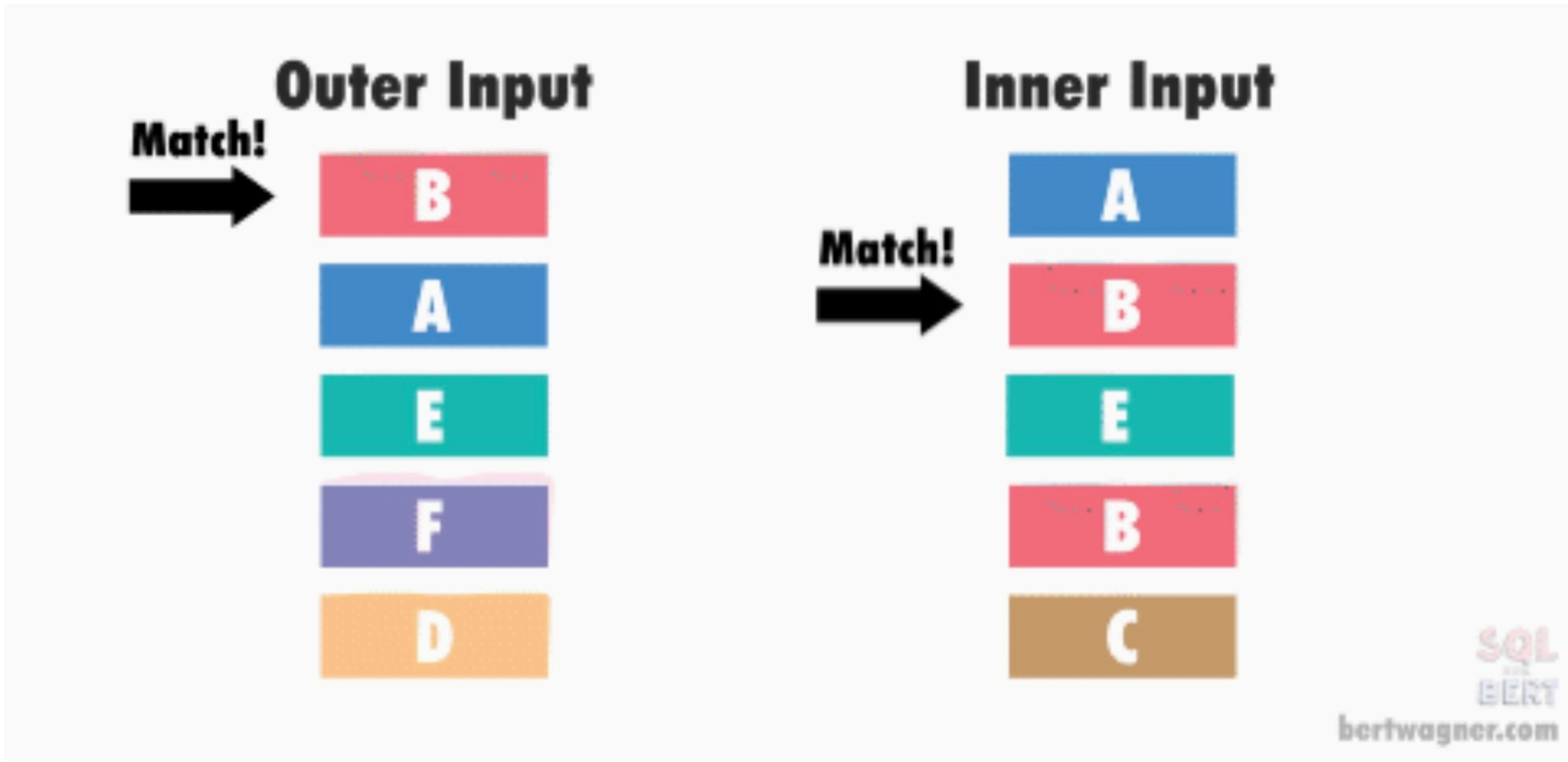
On `JOIN` Implementation

- The **`JOIN` operation**, between tables R and S on A and B attributes, is one of the most **time-consuming** operations in query processing.
- These are four of the most common techniques for performing a `JOIN`:
  1. Nested-loop Join
  2. Single-loop Join
  3. Sort-merge Join
  4. Hash-based Join
- Depending on the statistics, **the optimizer chooses** the approach with **the lowest estimated cost**.

Nested-loop Join

- The table R is set as the *outer table* and the table S is set as the *inner table*.
- For each row in the *outer table* that matches the single-table predicates (the filter F_R), the method retrieves all rows (nested-loop) in the *inner table* that satisfy the join predicate (F_J).
- The cost depends on the assigned role to R and S, the presence of index, and the size of the buffer.

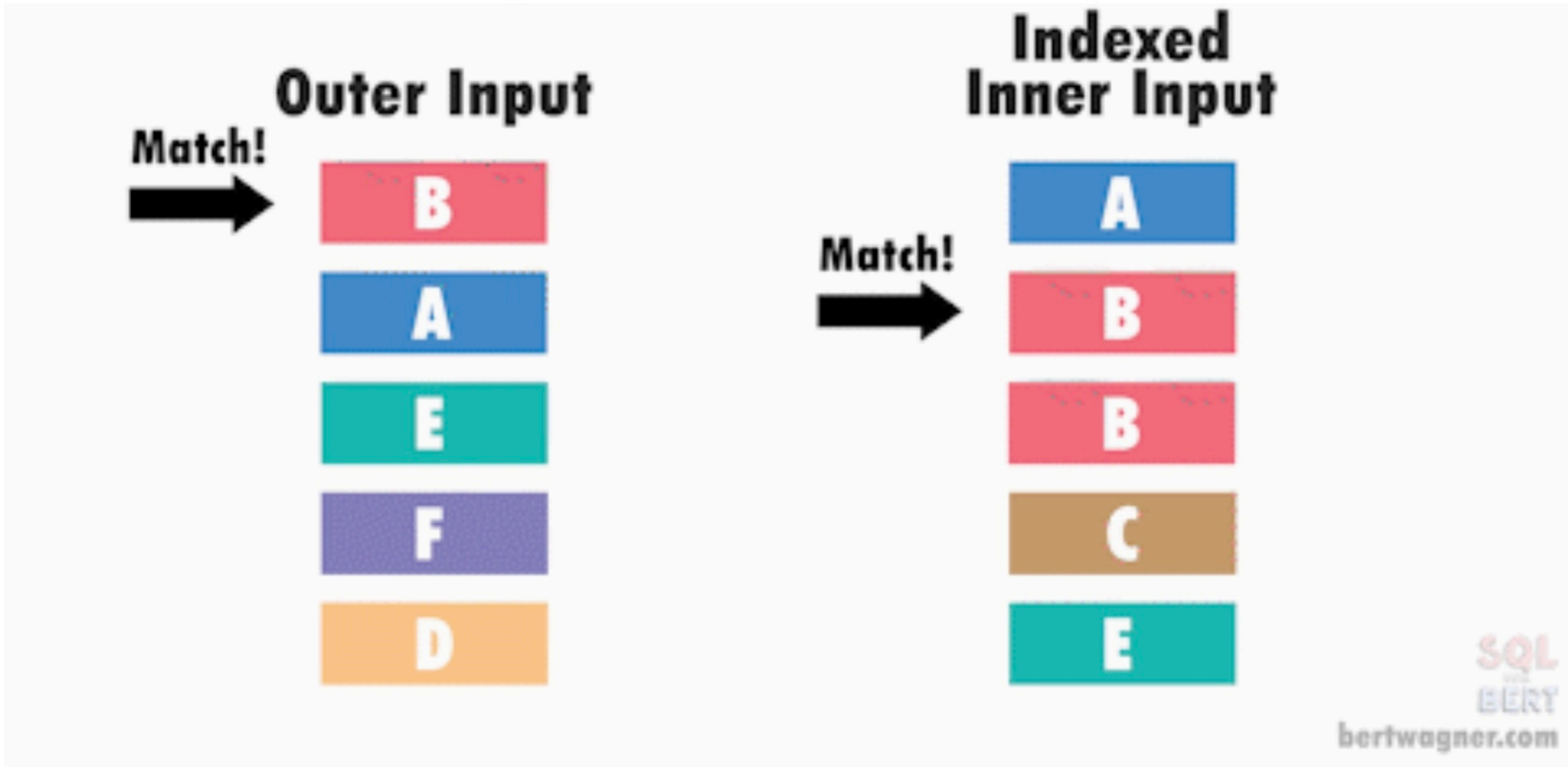
Visualizing Nested-loop Join



Single-loop Join

- An index (or hash key) exists for one of the two join attributes, for example attribute A of R.
- The method retrieves each row S, one at a time (single-loop), and then uses the access structure to retrieve directly all matching rows from R that satisfy the join predicate.

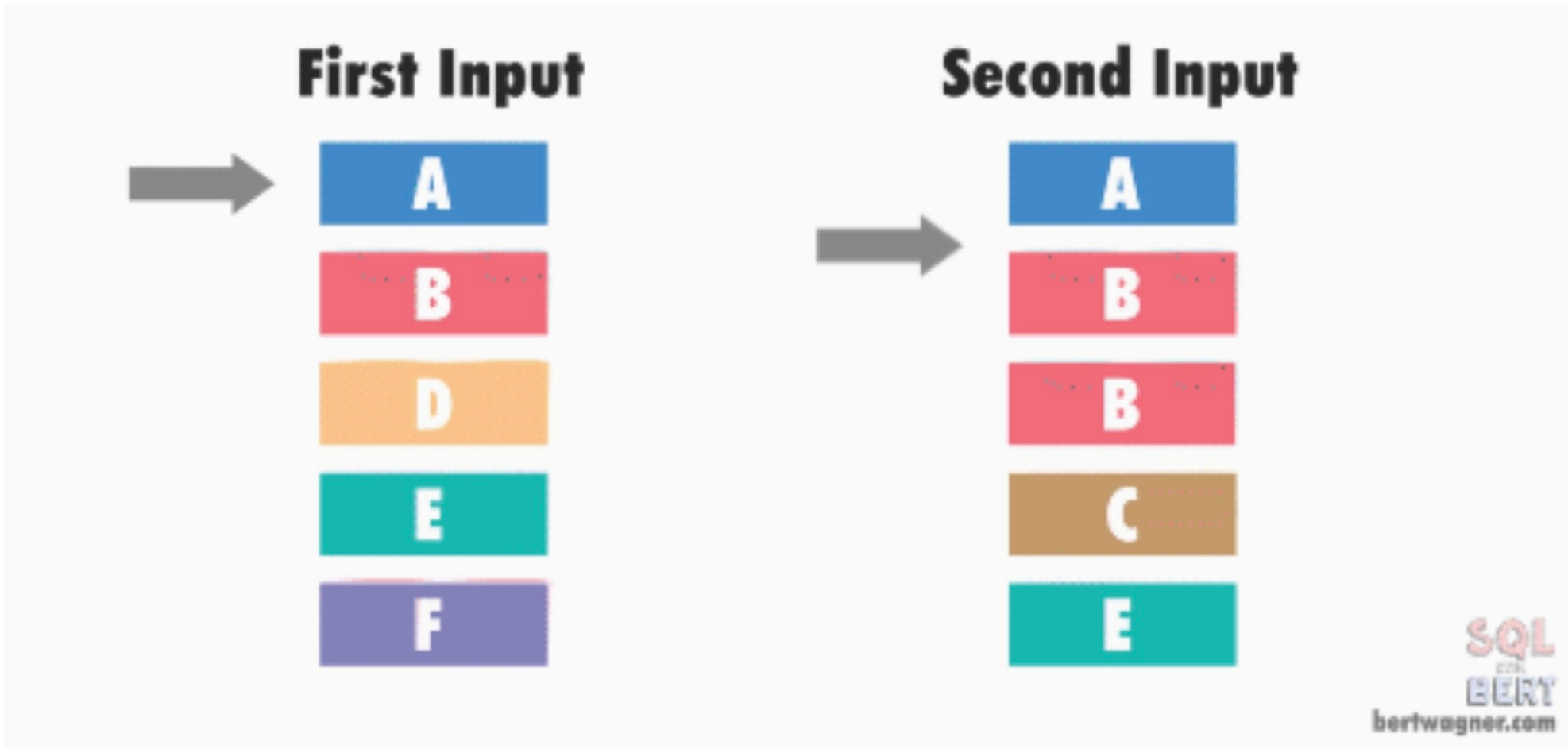
Visualizing Single-loop Join



Sort-merge Join

- The R and S tables are physically sorted by value of the join attributes A and B, respectively.
- Both tables are scanned (only once) in order of the join attributes, matching the records that have the same values for A and B.
- The optimizer may choose a sort-merge join for joining large amounts of data when join condition between two tables is not an equi-join (e.g., <, ≤, >, ≥) or sort is required by other operations.

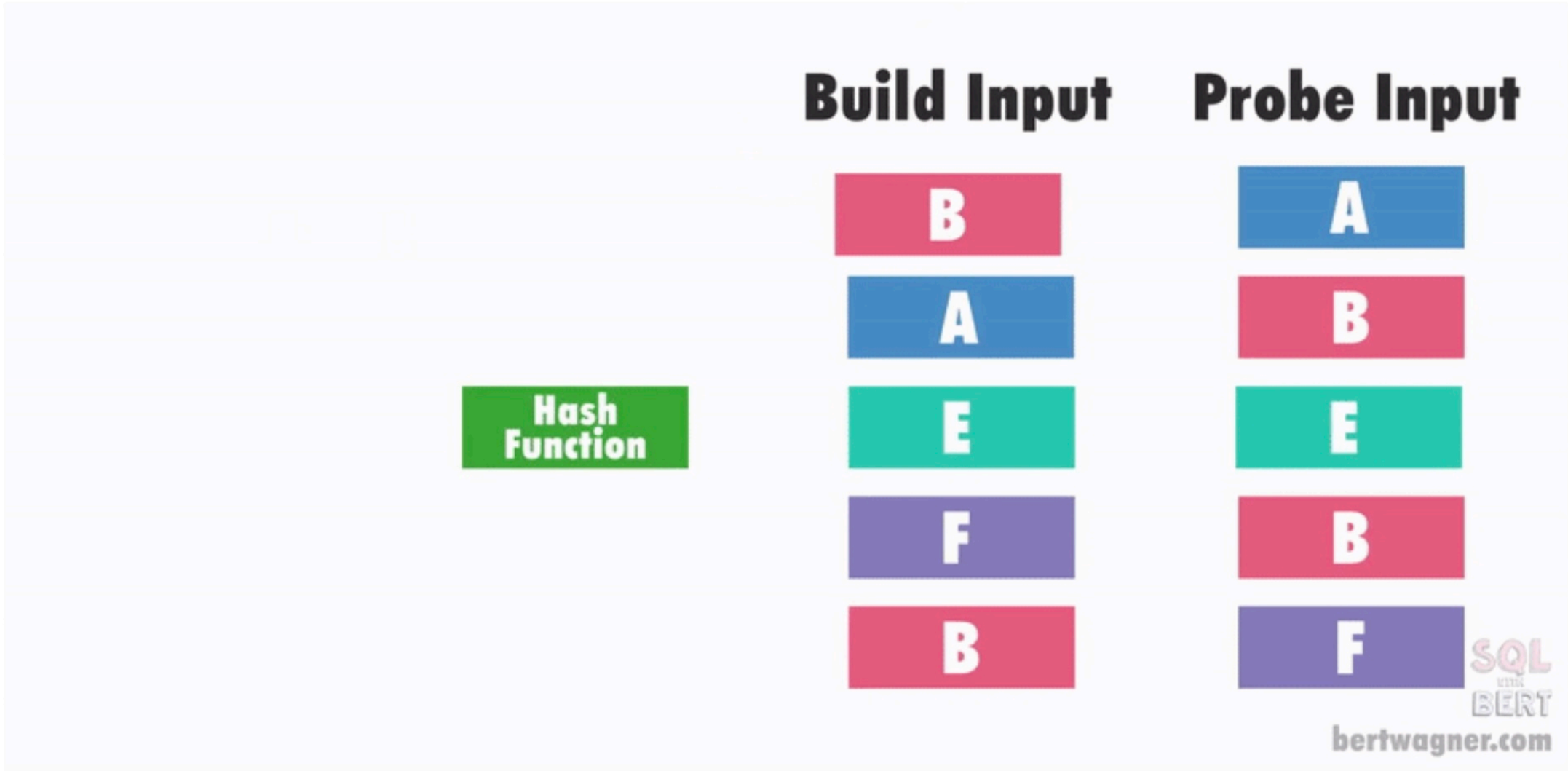
Visualizing Sort-merge Join



Hash-based Join

- The tables R and S are both hashed using the same hashing function on the join attributes.
- The new hash tables, each with *B* buckets, allow to find quicker a given join attribute's rows than by scanning the original relation. The final result can be obtained with B simple joins among the B buckets.
- Hash joins are typically more efficient for large tables.

Visualizing Hash-based Join



Exercise 2

Given the following schema:

LECTURER(<u>Id</u>, Surname, Dept)

STUDENT(<u>Id</u>, Surname)

COURSE(<u>Code</u>, Name)

EDITION(<u>Course</u>, <u>Year</u>, Lecturer)

EXAM(<u>Student</u>, <u>Course</u>, Year)

Write a SQL query that returns the surnames of those lecturers that have taught in courses for which at least 10 students passed the exam.

Exercise 2 - step 1.1

Let's highlight the involved entities:

LECTURER(Id, Surname, Dept)

STUDENT(Id, Surname)

? **COURSE**(Code, Name)

? **EDITION**(Course, Year, Lecturer)

EXAM(Student, Course, Year)

Write a SQL query that returns the surnames of those **lecturers** that have taught in **courses** for which at least 10 students passed the **exam**.

Exercise 2 - comments

The following passage within the text is quite ambiguous:

"*... that have taught in courses ...*"

Which relation shall we use: `COURSE` or `EDITION`?

The `EXAM` relation has informations on both `Course` and `Year`

`EXAM(`<u>`Student`</u>`,` <u>`Course`</u>`,` `Year)`

so we are going to use `EDITION`.

Exercise 2 - step 1.2

Write a SQL query that returns the surnames of those **lecturers** that have taught in "**courses of any edition**" for which at least 10 students passed the **exam**.

- Which is the desired result?

```
SELECT DISTINCT L.Surname
FROM LECTURER AS L, EDITION AS C,
     EXAM AS E
```

Exercise 2 - step 1.3

Which conditions allow to filter out the data?

- Let's connect the *foreign* and *primary keys*:

```
WHERE L.Id = C.Lecturer AND
      C.Course = E.Course AND
      C.Year = E.Year
```

- We aggregate the exams by lecturer, and we filter them by the number of the passed exam:
  - The lecturer's `Id` is a number, but we have to aggregate `Surname` too because it must appear in the result.
  - We must aggregate both by lecturer and by the courses because we want the number of exam per course and not per lecturer.

Exercise 2 - solution

```
SELECT DISTINCT L.Surname
FROM LECTURER AS L, EDITION AS C,
     EXAM AS E
WHERE L.Id = C.Lecturer AND
      C.Course = E.Course AND
      C.Year = E.Year
GROUP BY L.Id, L.Surname, C.Course, C.Year
HAVING COUNT(*) > 10
```

- What happen if we remove `C.Year` from `GROUP BY` clause?
- What happen if we remove `C.Course` and `C.Year` from `GROUP BY` clause?

Exercise 3

Given the following schema:

TRAIN(<u>Code</u>, Departure, Arrival, Mile)

Write a SQL query that returns the minimum and the maximum length of the routes between Boston and Chicago without interchanges.

Exercise 3 - comments

- "... without interchanges." means that we only have to consider direct routes and we have to filter tuples on `Departure` and `Arrival`.

- Moreover, we have to use `MIN()` and `MAX()` *SQL aggregation functions* for retrieving the tuples with the minimum and the maximum trip length.

Exercise 3 - solution

```
SELECT MIN(T.Mile) AS MinLength,
       MAX(T.Mile) AS MaxLength
FROM TRAIN AS T
WHERE T.Departure = “Boston” AND
      T.Arrival = “Chicago”
```

Exercise 4

Given the following schema:

```
REGION(Name, Population, Surface)
RESIDENCE(Id, Surname, Region)
```

Write a SQL query that returns the regions having more inhabitants than residents.

Exercise 4 - a piece of the solution

We can obtain the number of citizen of a region by aggregating RESIDENCE by `Region` (i.e., the name).

```
SELECT Region, COUNT(*) AS Citizen
FROM RESIDENCE
GROUP BY Region
```

Thus, we can define a **view** that could be used in a `FROM` clause, in order to compare the result with the `Population` value.

Exercise 4 - final solution

```
SELECT A.Name
FROM REGION AS A,
     (SELECT Region, COUNT(*) AS Citizen
      FROM RESIDENCE
      GROUP BY Region) AS C
WHERE A.Name = C.Region AND
      A.Population > C.Citizen
```

Exercise 4 - optimized solution

```
CREATE VIEW C AS
SELECT Region, COUNT(*) AS Citizen
FROM RESIDENCE
GROUP BY Region
```

```
SELECT A.Name
FROM REGION AS A, C
WHERE A.Name = C.Region AND
      A.Population > C.Citizen
```

Exercise 5

RATING

FilmID	UserID	Rating
11234	A984	7
21234	A984	9
31234	B563	8

USER

UserID	Alias	Age
A984	JamesSmith	20
B563	MaryJohnson	15
F123	JohnBrown	36

FILM

FilmID	Title	Year	Director
21234	Blade Runner	1982	Ridley Scott
41234	Pulp Fiction	1994	Quentin Tarantino
11234	Dead Poets Society	1989	Peter Weir

Write a SQL query that returns the titles of those films produced between 1990 and 2000, having an average rating greater than or equal to 7.

Exercise 5 - solution

```
SELECT FILM.Title
FROM RATING NATURAL JOIN FILM
WHERE FILM.Year ≥ 1990 AND
      FILM.Year ≤ 2000
GROUP BY FILM.FilmID, FILM.Title
HAVING AVG(RATING.Rating) ≥ 7
```

Exercise 6

RATING		
FilmID	UserID	Rating
11234	A984	7
21234	A984	9
31234	B563	8

USER		
UserID	Alias	Age
A984	JamesSmith	20
B563	MaryJohnson	15
F123	JohnBrown	36

FILM			
FilmID	Title	Year	Director
21234	Blade Runner	1982	Ridley Scott
41234	Pulp Fiction	1994	Quentin Tarantino
11234	Dead Poets Society	1989	Peter Weir

Write a SQL query that returns the minimum age of users who rated “Blade Runner” greater than 8.

Exercise 6 - solution

```
SELECT MIN(USER.Age)
FROM RATING NATURAL JOIN USER
            NATURAL JOIN FILM
WHERE FILM.Title = “Blade Runner” AND
      RATING.Rating > 8
```

Exercise 7

SONG

ID	Title	Album	Plays	Length
234	Tennessee Whiskey	4444	314	293
112	Purple Rain	3333	111	245
324	Shackled and Drawn	1111	674	226

PLAYLIST

Name	Position	Song
Summer	1	546
Summer	2	315
Party	1	234

ALBUM

ID	Title	Year	Artist
1111	Wrecking Ball	2012	234
2222	Sunshine on Leith	1988	657
3333	Purple Rain	1984	756

ARTIST

ID	Name	Labels
234	Bruce Springsteen	Columbia
112	Sting	UMG
324	Chris Stapleton	WCM

Write a SQL query that returns the name of those playlists having at least one song from an album published before 2001 by an UMG artist.

Exercise 7 - solution

```
SELECT DISTINCT PLAYLIST.Name
FROM PLAYLIST JOIN SONG
              ON PLAYLIST.Song = SONG.ID
                 JOIN ALBUM
                 ON SONG.Album = ALBUM.ID
                    JOIN ARTIST
                    ON ALBUM.Artist = ARTIST.ID
WHERE ALBUM.Year < 2001 AND
      ARTIST.Label = “UMG”
```

Another Schema

ACTOR(Id, Name, Year, Nationality)

RECITAL(Actor, Film)

FILM(Code, Title, Year, Director, Country, Genre)

SCREENING(Code, Film, Room, Date, Profits)

ROOM(Code, Name, Seats, City)

Exercise 8

Write a SQL query that returns the title of Fellini's films produced after 1960.

```
SELECT F.Title
FROM FILM AS F
WHERE F.Director = “Fellini” AND
      F.Year > 1960
```

Exercise 9

Write a SQL query that returns the title of Japanese science fiction films produced after 1990 or French science fiction films.

```
SELECT F.Title
FROM FILM AS F
WHERE F.Genre = “Sci-fi” AND
      ((F.Country = “Japan” AND
        F.Year > 1990) OR
       F.Country = “France”)
```

Exercise 10

Write a SQL query that returns the title and the genre of films screened in London on last Christmas Day.

```
SELECT DISTINCT F.Title, F.Genre
FROM FILM AS F, SCREENING AS S, ROOM AS R
WHERE S.Date = 2018-12-25 AND
      R.City = “London” AND
      F.Code = S.Film AND
      S.Room = R.Code
```

Exercise 11

Write a SQL query that returns the number of screening room in London with more than 60 seats.

```
SELECT COUNT(*)
FROM ROOM AS R
WHERE R.City = “London” AND
      R.Seats > 60
```

Exercise 12

Write a SQL query that returns for each pair of director and actor their names, and the number of films where they have worked together.

```
SELECT F.Director, A.Name,
       COUNT(*) AS NumMovies
FROM ACTOR AS A, RECITAL AS R, FILM AS F
WHERE A.Id = R.Actor AND
      R.Film = F.Code
GROUP BY F.Director, A.Id, A.Name
```

Exercise 13

Write a SQL query that returns the director and the title of films in which less than 6 actors act.

```
SELECT F.Director, F.Title
FROM FILM AS F, RECITAL AS R
WHERE F.Code = R.Film
GROUP BY F.Code, F.Title, F.Director
HAVING COUNT(*) < 6
```

What about of films without actors?

Exercise 13 - a better solution

Write a SQL query that returns the director and the title of films in which less than 6 actors act.

```
SELECT F.Director, F.Title
FROM FILM AS F
WHERE 6 > (SELECT COUNT(*)
           FROM RECITAL AS R
           WHERE F.Code = R.Film)
```

Exercise 14

Write a SQL query that returns the name and the whole takings of those rooms in Rome that in January 2005 have gained more than $20'000.

```
SELECT R.Name, SUM(S.Profits)
FROM ROOM AS R, SCREENING AS S
WHERE R.Code = S.Room AND
      R.City = "Rome" AND
      2005-01-01 < S.Date AND
      S.Date < 2005-01-31
GROUP BY R.Code, R.Name
HAVING SUM(S.Profits) > 20'000
```

Exercise 15

Write a SQL query that returns the title of films that have never been screened in Berlin.

```
SELECT F.Title
FROM FILM AS F
WHERE NOT EXISTS (SELECT *
                  FROM SCREENING AS S, ROOM AS R
                  WHERE R.City = “Berlin” AND
                        F.Code = S.Film AND
                        S.Room = R.Code)
```

Exercise 15 - another solution

Write a SQL query that returns the title of films that have never been screened in Berlin.

```
SELECT F.Title
FROM FILM AS F
WHERE “Berlin” NOT IN (SELECT R.City
                       FROM SCREENING AS S,
                            ROOM AS R
                       WHERE F.Code = S.Film AND
                             S.Room = R.Code)
```

Exercise 16

Write a SQL query that returns the title of films that have never had a screening with revenues greater than $500.

```
SELECT F.Title
FROM FILM AS F
WHERE NOT EXISTS (SELECT *
                  FROM SCREENING AS S
                  WHERE S.Profits > 500 AND
                        F.Code = S.Film)
```

Exercise 17

Write a SQL query that returns the title of films that have always earned more than $500 in all their screening.

```
SELECT F.Title
FROM FILM AS F
WHERE 500 ≤ (SELECT MIN(S.Profits)
             FROM SCREENING AS S
             WHERE F.Code = S.Film)
```

One More Schema

MUSEUM(<u>Name</u>, City)

ARTIST(<u>Name</u>, Nationality)

WORK(<u>Code</u>, Title, NameM, NameA)

CHARACTER(<u>Name</u>, <u>CodeW</u>)

Exercise 18

Write a SQL query that returns the name of the museums in London that do not have Tiziano's works.

```
SELECT M.Name
FROM MUSEUM AS M
WHERE M.City = “London” AND
      NOT EXISTS (SELECT *
                  FROM WORK AS W
                  WHERE W.NameA = “Tiziano”
                        AND M.Name = W.NameM)
```

Exercise 19

Write a SQL query that returns the name of the museums in London that only have Tiziano's works.

```
SELECT M.Name
FROM MUSEUM AS M
WHERE M.City = “London” AND
      “Tiziano” = ALL (SELECT W.NameA
                       FROM WORK AS W
                       WHERE M.Name = W.NameM)
```

Exercise 20

Write a SQL query that returns the name of the museums that have at least 20 works by Italian artists.

```
SELECT W.NameM AS Museum
FROM WORK AS W, ARTIST AS A
WHERE W.NameA = A.Nome AND
      A.Nationality = “Italy”
GROUP BY W.NameM
HAVING COUNT(*) ≥ 20
```

Try it Yourself

https://www.w3schools.com/sql/

https://www.w3resource.com/sql-exercises/