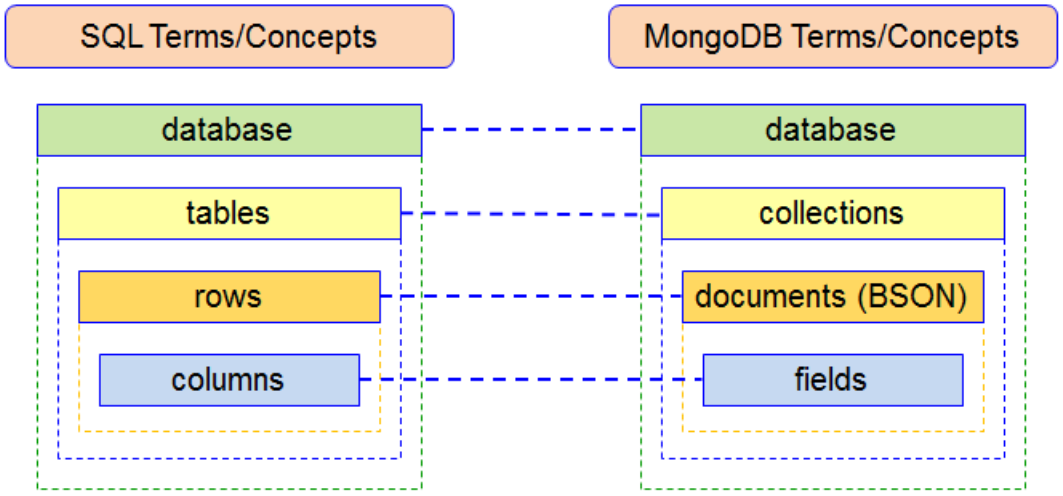
Lab 5 - SQL


We won't be able to deliver our product in time because of some issue with MySQL...
WHAT??!
THEN USE SOMEBODY ELSE'S SQL, BUT I WANT THE PRODUCT IN TIME.
Brainstuck.com

DATABASES

A database is a collection of information that is organized so that it can be easily accessed, managed and updated



SQL Server, Oracle, Mysql etc

MongoDB, CoucheDB, Cassandra, HBase

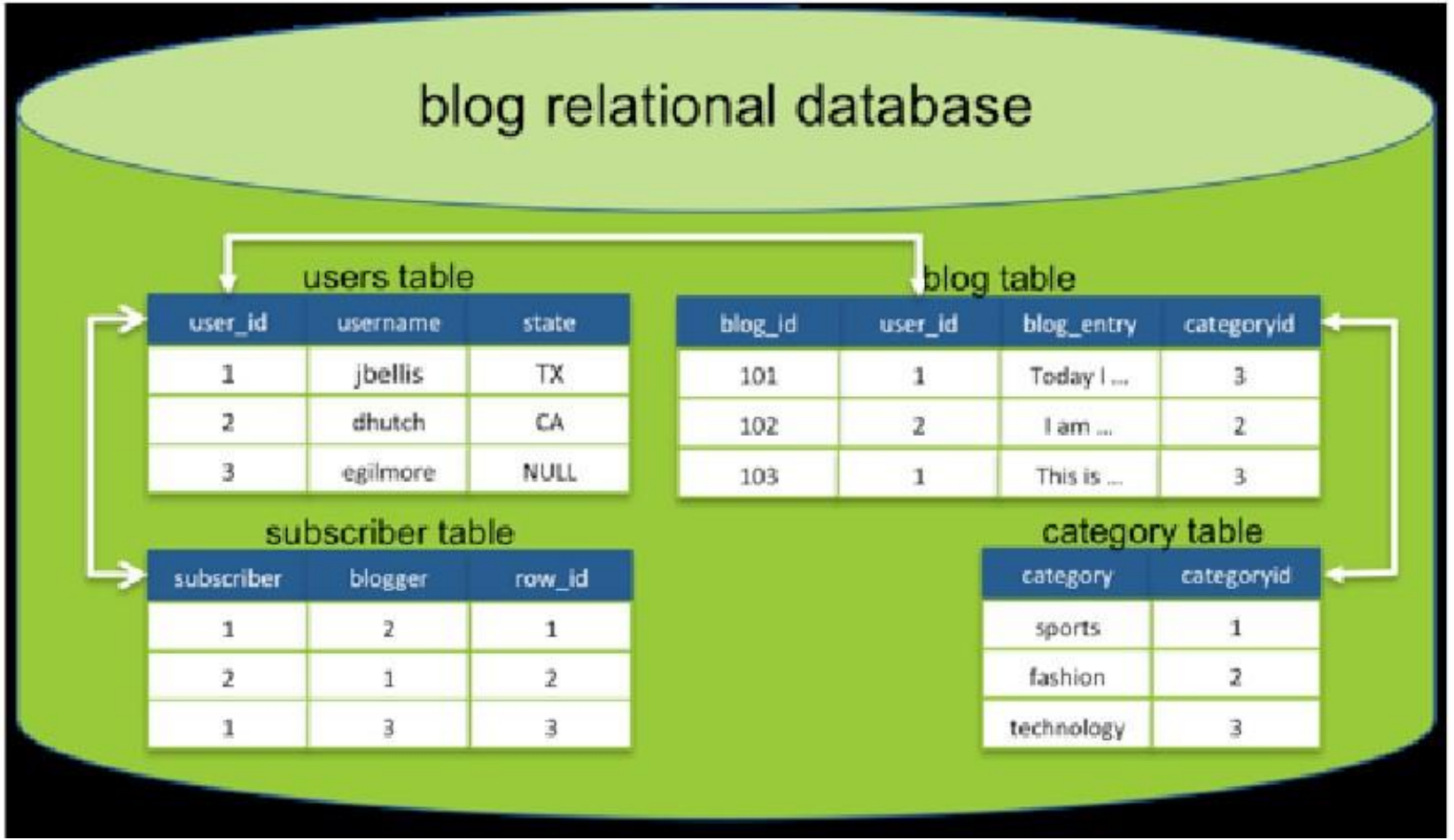
Modified Material by Liz Boese

RELATIONAL DATABASES

- A relational database contains one or more tables of information.
- The rows in a table are called **records** and the columns in a table are called **attributes** or fields.

Relational Databases

- **Relational Databases** store data according to a set of defined **entities** and **relations**
- **Entities** (tables) consist of related **attributes** (fields)



Entity Data Stored in Tables

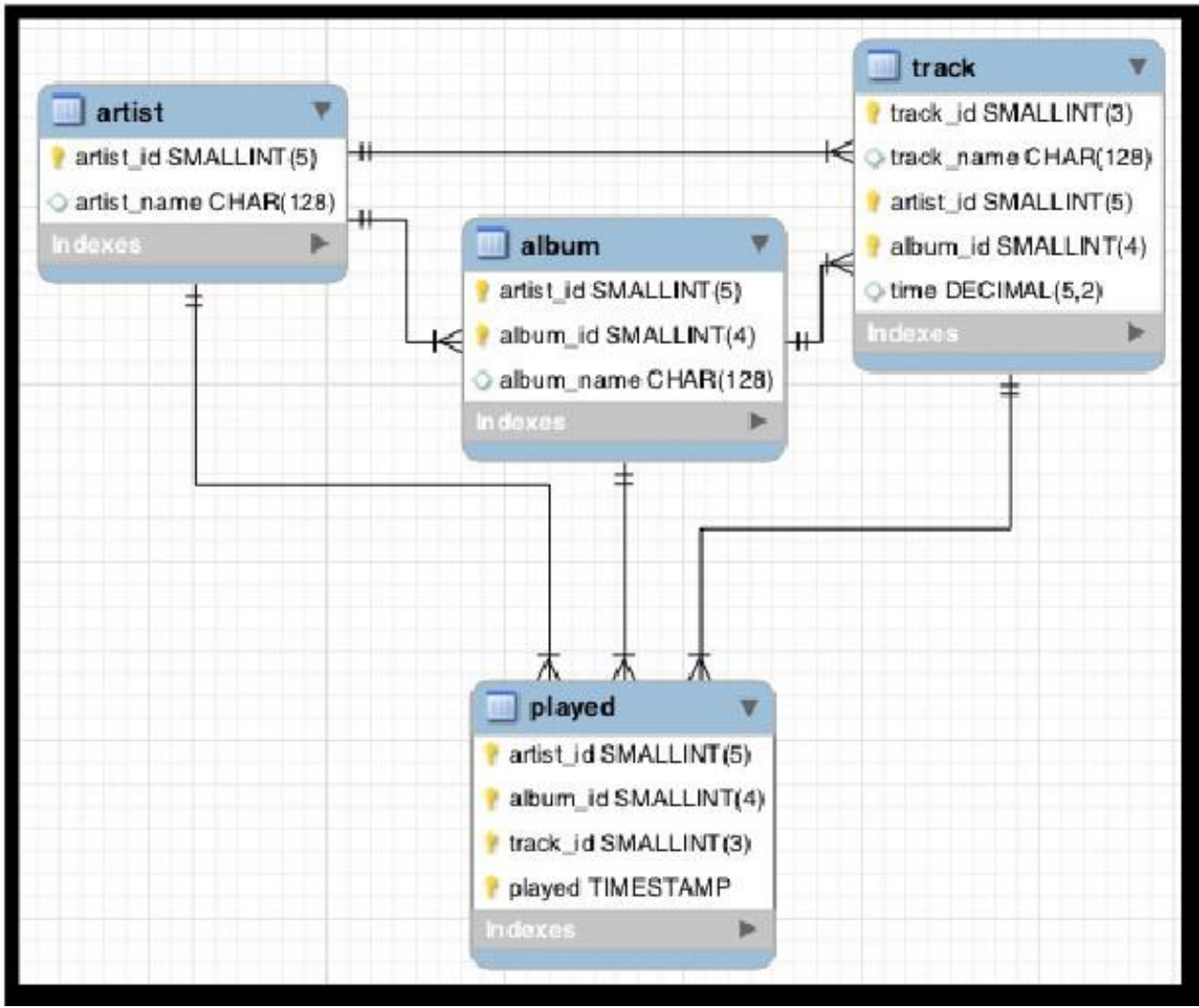
- Table columns contain attributes
- Table rows (tuples) contain items

Table: Category

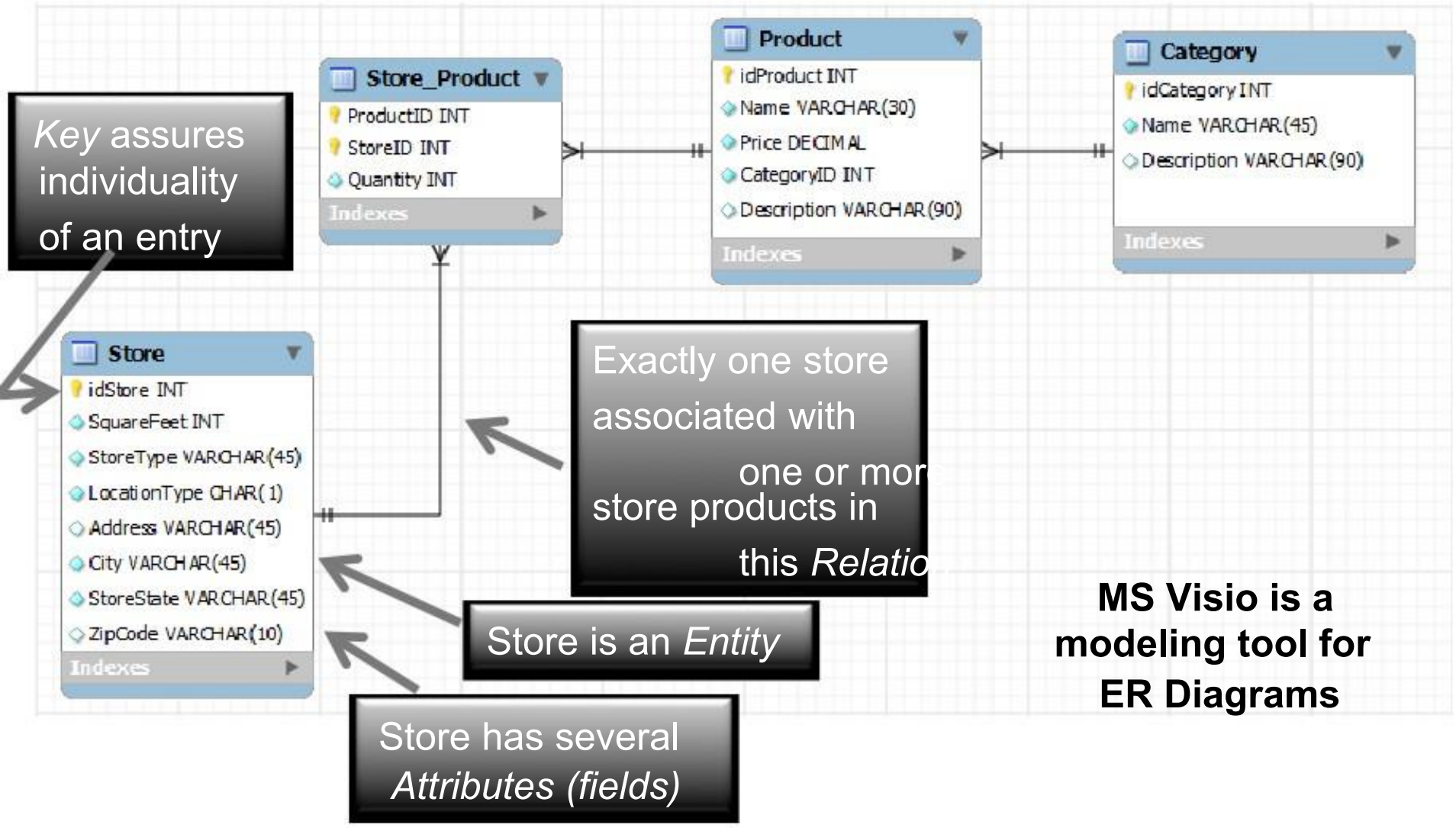
ID Category	Name	Description
1	Auto Parts	Things to service a car
2	Electronics	TVs, DVD players, etc.

ER Diagrams

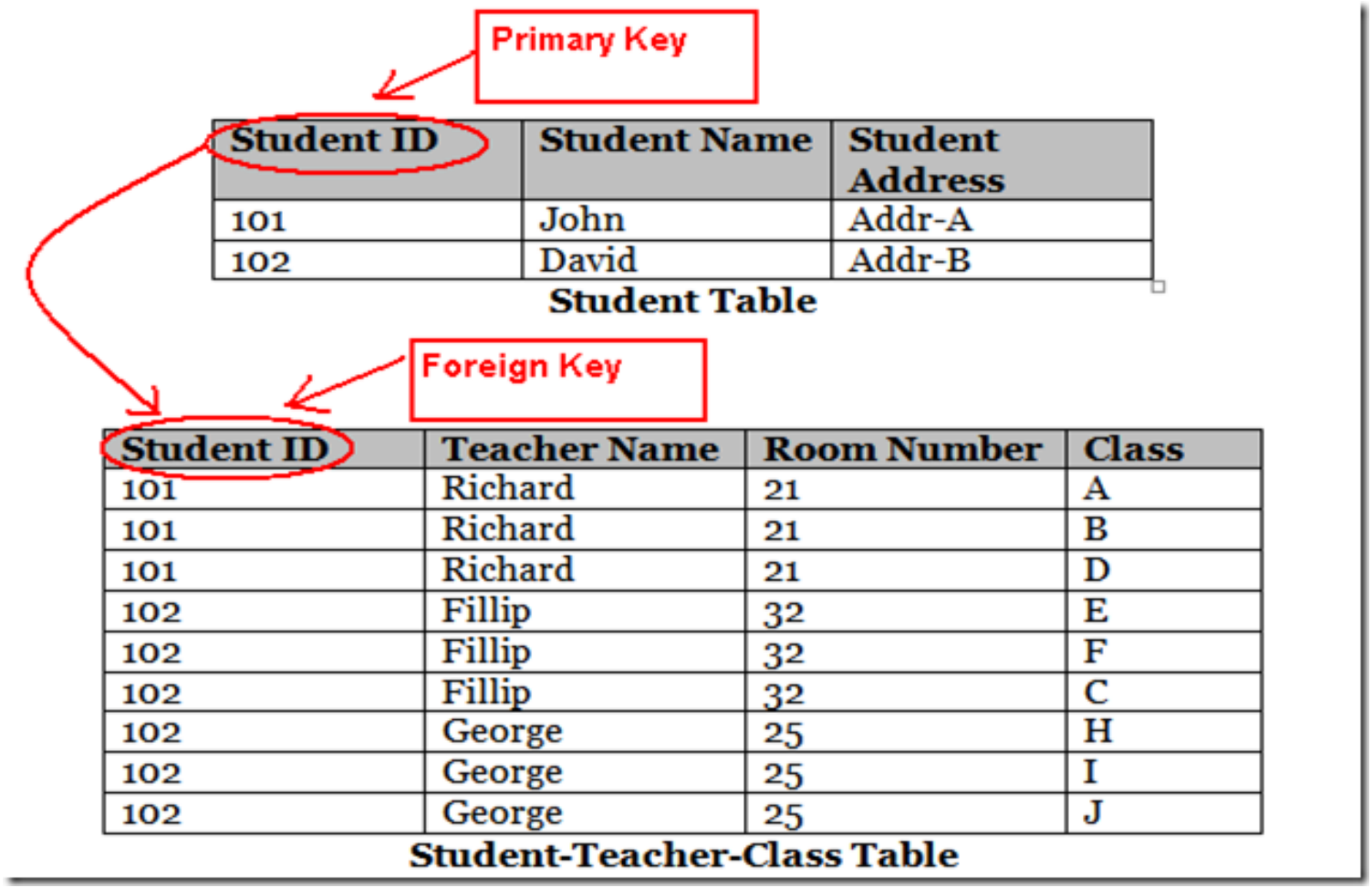
- **ER Diagrams** support modeling and analysis of database relational structure
  - **Relations** depict how two entities are related: "*X entity contains zero or more of Y entity*"



ER Diagram



MS Visio is a modeling tool for ER Diagrams



Student ID	Student Name	Student Address
101	John	Addr-A
102	David	Addr-B

Student Table

Student ID	Teacher Name	Room Number	Class
101	Richard	21	A
101	Richard	21	B
101	Richard	21	D
102	Fillip	32	E
102	Fillip	32	F
102	Fillip	32	C
102	George	25	H
102	George	25	I
102	George	25	J

Student-Teacher-Class Table

E-R Diagrams

- There are variations to ER Diagrams
- For this course we are focused on the ones we discussed. This is based on the diagrams that are usually auto-produced for databases.
  - Designate keys
  - Attributes inside box *(not ovals outside of it)*
- You need to be able to read an ER Diagram as you may have one to work with for coding
- Tools for E-R Diagram creation: MS-Visio, MySQL Workbench, SQL Management studio etc.,

Structured Query Language

SQL

ID	First Name	Last Name	Email	Year of Birth
1	Peter	Lee	plee@university.edu	1992
2	Jonathan	Edwards	jedwards@university.edu	1994
3	Marilyn	Johnson	mjohnson@university.edu	1993
6	Joe	Kim	jkim@university.edu	1992
12	Haley	Martinez	hmartinez@university.edu	1993
14	John	Mfume	jmfume@university.edu	1991
15	David	Letty	dletty@university.edu	1995

Table: Students

- How do I get only the first names of all the students?
- How do I get details of a student who's last name is Kim?

Select Basics

- **SQL** is a language to write queries over the data
- Most basic of all SQL statements

SELECT *columnlist* **FROM** *table*

§ *Columnlist* named by comma separated list.

SELECT firstName, lastName FROM Students

§ Use * as a shortcut for all columns.

SELECT * FROM Students

All caps for SQL keywords is a common convention

§ **Aliasing** can be used to change the column name

- This is useful to improve readability of reports

SELECT data3 AS start_date FROM Students

Filtering and Ordering

```
SELECT columnlist
FROM table
WHERE columnCondition
ORDER BY columnOrder [ASC/DESC]
```

- **WHERE** clause filters based on columns using Boolean and logical operators

```
SELECT * FROM Students
WHERE lastName = "Kim"
```

- **ORDER BY** defaults to **ASC**

```
SELECT * FROM Students
ORDER BY lastName DESC
```

Categories

CategoryID	Category Name
10	Fruits
20	Vegetables

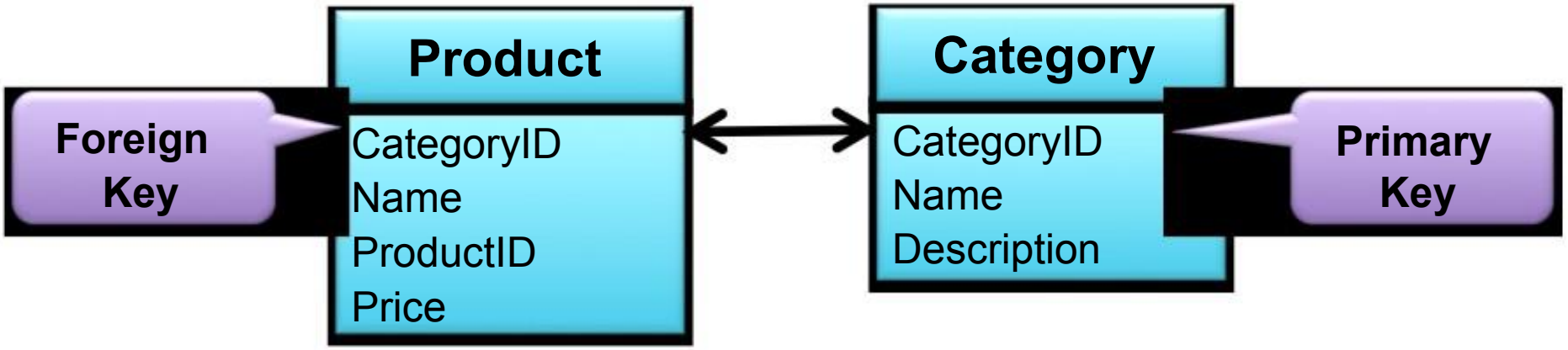
Products

ProductID	ProductName	Price	CategoryID
1	Apples	0.40	10
2	Oranges	1.10	10
3	Lettuce	0.60	20
4	Squash	1.20	20

How to get a list of products along with their category names ?

Joining tables

- Relationships are generally modeled through "key" columns.
- Primary and Foreign keys
- Example - "CategoryID" column in the Product table references the "id" column in the Category table.



Joining Tables using Where

- Get results from multiple tables using where

```
SELECT columnlist
FROM table1, table2
WHERE table1.col_1 = table2.col_2
```

- Example

```
SELECT name, price
FROM Product, Category
WHERE Category.CategoryID = Product.CategoryID
```

Join Syntax

- Get results from multiple tables

```
SELECT columnlist
FROM table1
JOIN table2 ON table1.col_1 = table2.col_2
```

- Example

```
SELECT name, price
FROM Product
JOIN Category ON CategoryID = id
```

Join Syntax

- WHERE vs JOIN?

INNER JOIN is ANSI syntax which you should use.
It is generally considered more readable, especially when you join lots of tables.

It can also be easily replaced with an OUTER JOIN whenever a need arises.

The WHERE syntax is more relational model oriented.

A result of two tables JOIN'ed is a cartesian product of the tables to which a filter is applied which selects only those rows with joining columns matching.

It's easier to see this with the WHERE syntax.

Also note that MySQL also has a STRAIGHT_JOIN clause.

Using this clause, you can control the JOIN order: which table is scanned in the outer loop and which one is in the inner loop.

You cannot control this in MySQL using WHERE syntax.

Name Resolution

```
SELECT name, price
FROM Product
JOIN Category ON categoryID = id
```

- Query engine must be able to resolve columns;
  in the example: “name” is ambiguous if there are columns called “name” in both tables.

- Fully qualified syntax can be used

  SELECT tableName.columnName, …

  Ex:

  SELECT Product.Name, Category.Name, …

Aliases

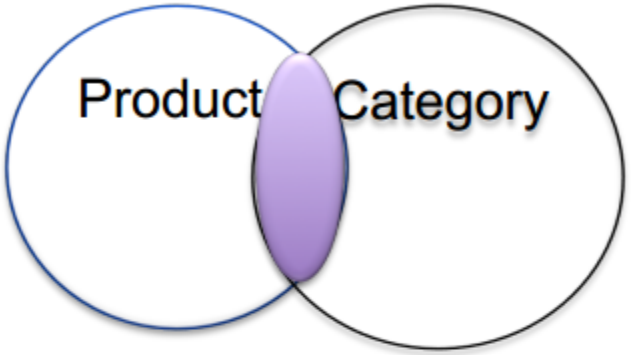
Aliases

- Shorten typing
- Rename columns.

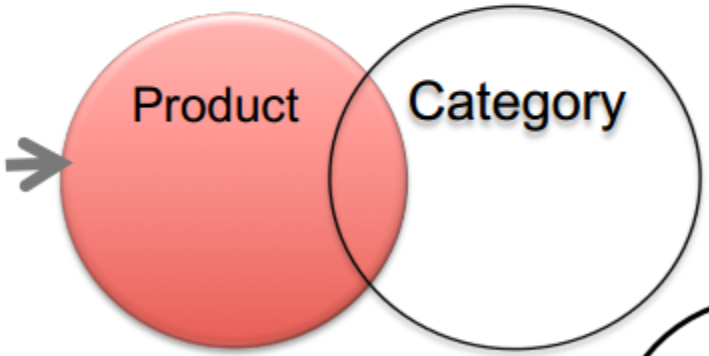
- On tables and/or column names
  - FROM Product p
- "AS" keyword optional
  - *(usually used on column names but not on table names)*
  - *p.name as "ProductName"*

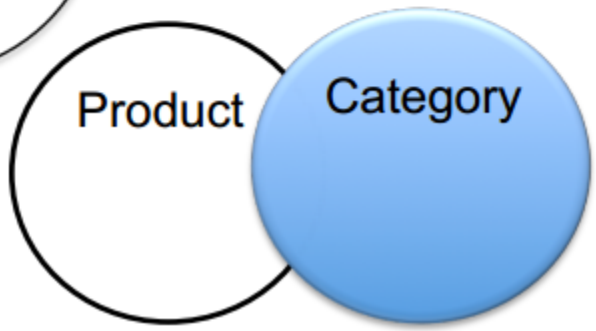
```
SELECT p.name as "ProductName", c.name
FROM Product p
JOIN category c ON p.CategoryID = c.CategoryID
```

JOIN Types



- **JOIN** is shortcut for **INNER JOIN**
  - Only records that match are returned.
- **OUTER JOIN** may return records that do not match the filter condition.
  - **LEFT JOIN, RIGHT JOIN** syntax signifies which table has "optional" values.





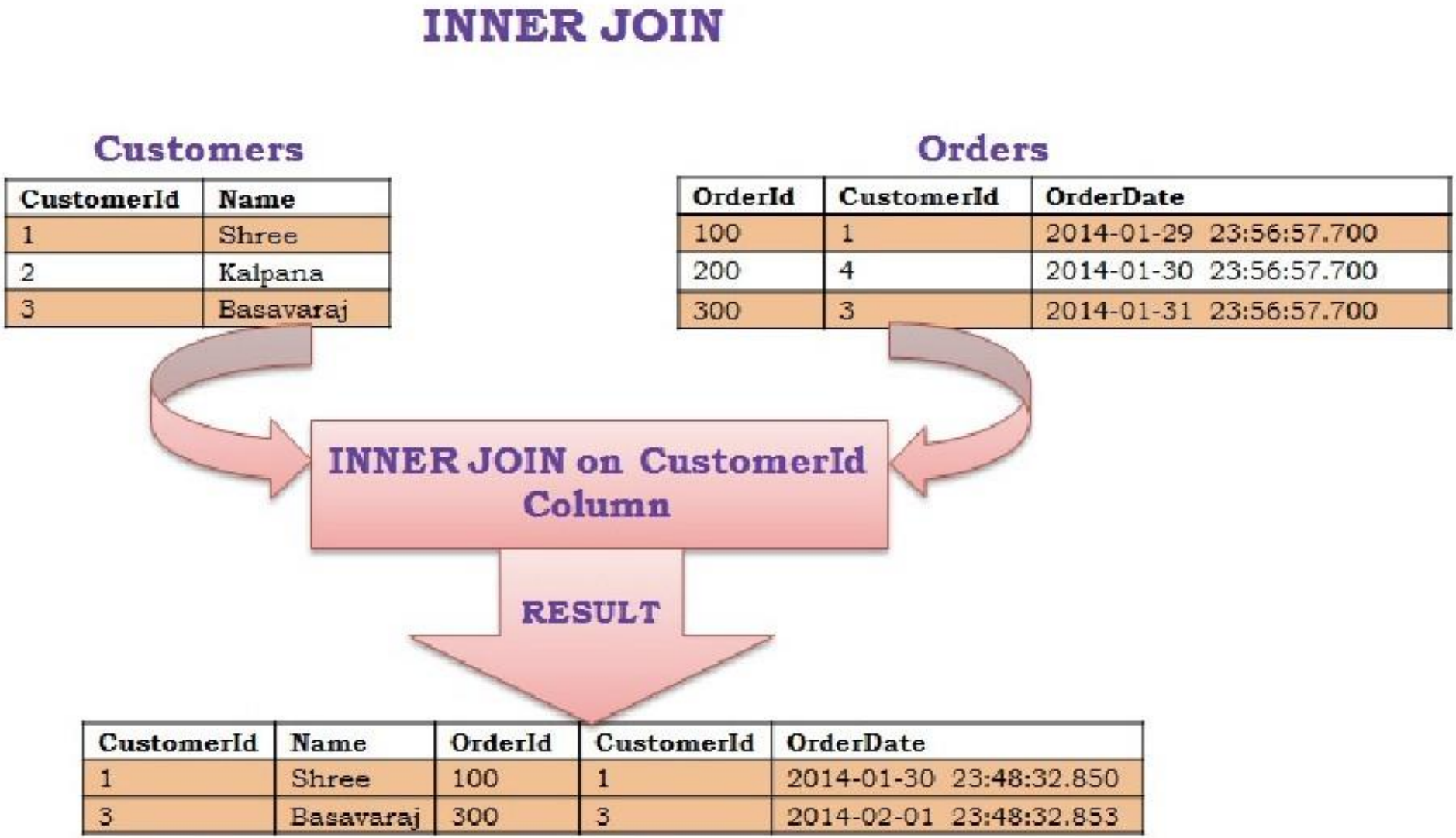
- Null values returned for non-matches.

Inner Join

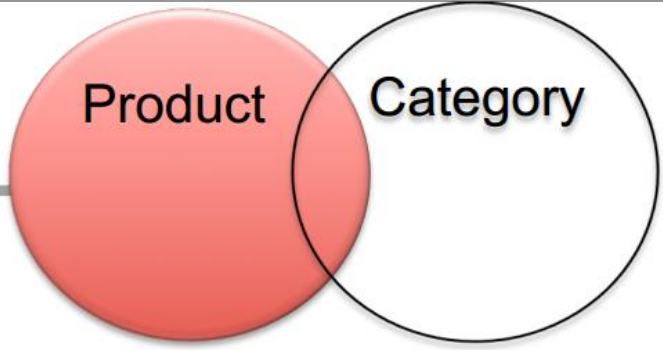
```
SELECT *
FROM   Customers C
          INNER JOIN Orders O
              ON O.CustomerId = C.CustomerId
```

Inner Join

```
SELECT *
FROM   Customers C
          INNER JOIN Orders O
              ON O.CustomerId = C.CustomerId
```



Left Join

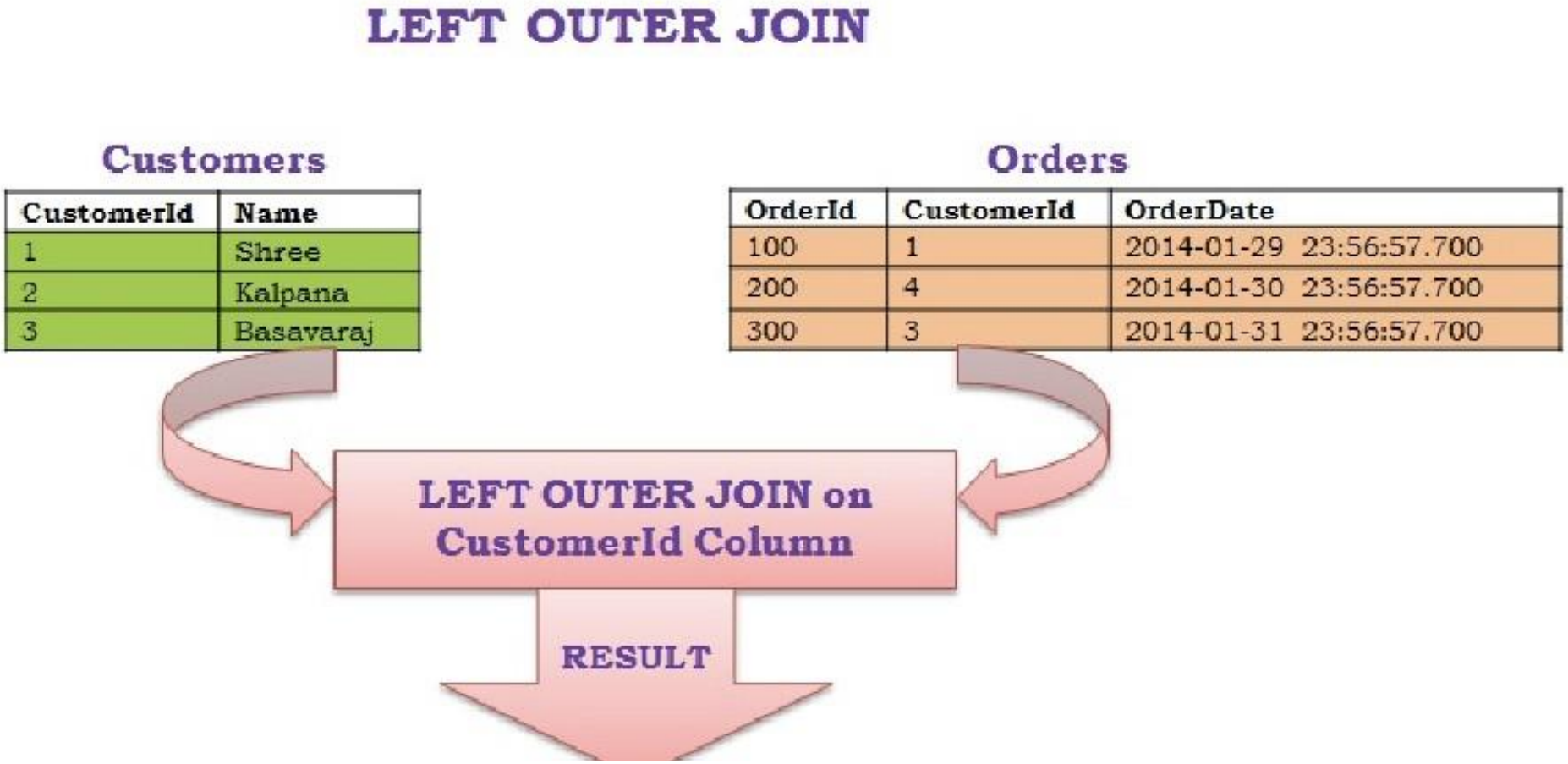


- Left join, sometimes called left outer join
- The LEFT JOIN keyword returns all the rows from the left table (Product), even if there are no matches in the right table (Category).
- *A list of all products and their categories even if a product not associated to a category.*
  If no products use a particular category, that category does not show up

```
SELECT p.name, c.name
FROM Product p
LEFT JOIN category c ON p.categoryID = c.id
```

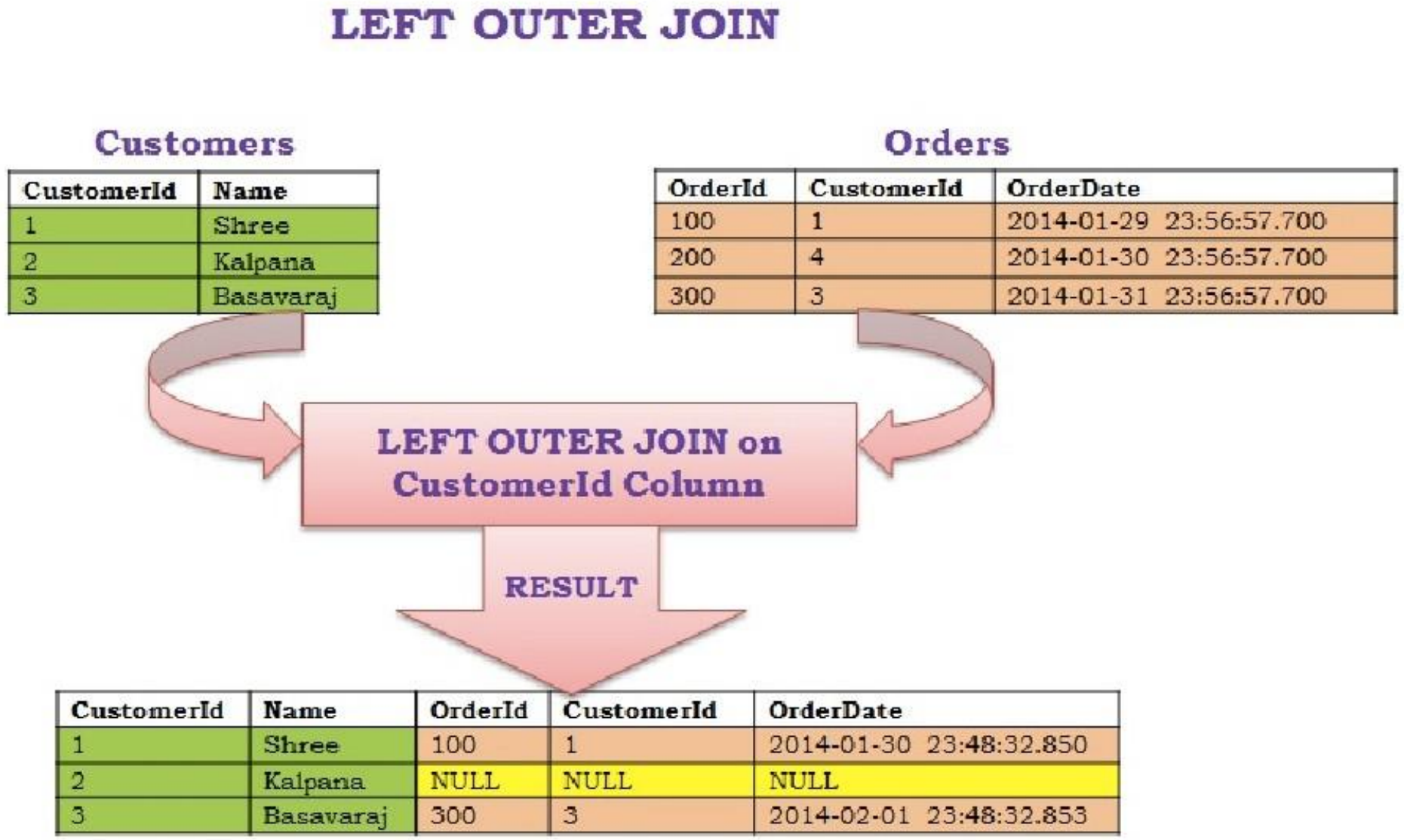
Left Join

```
SELECT *
FROM   Customers C
       LEFT OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```

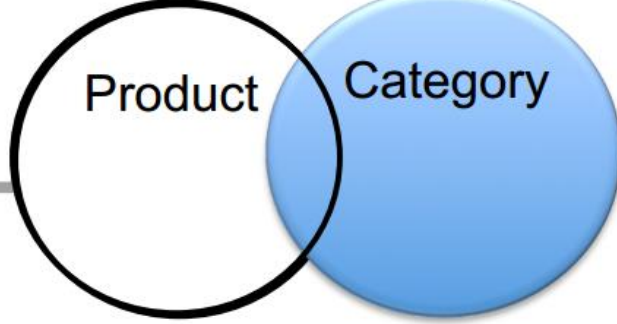


Left Join

```
SELECT *
FROM   Customers C
       LEFT OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```



Right Join



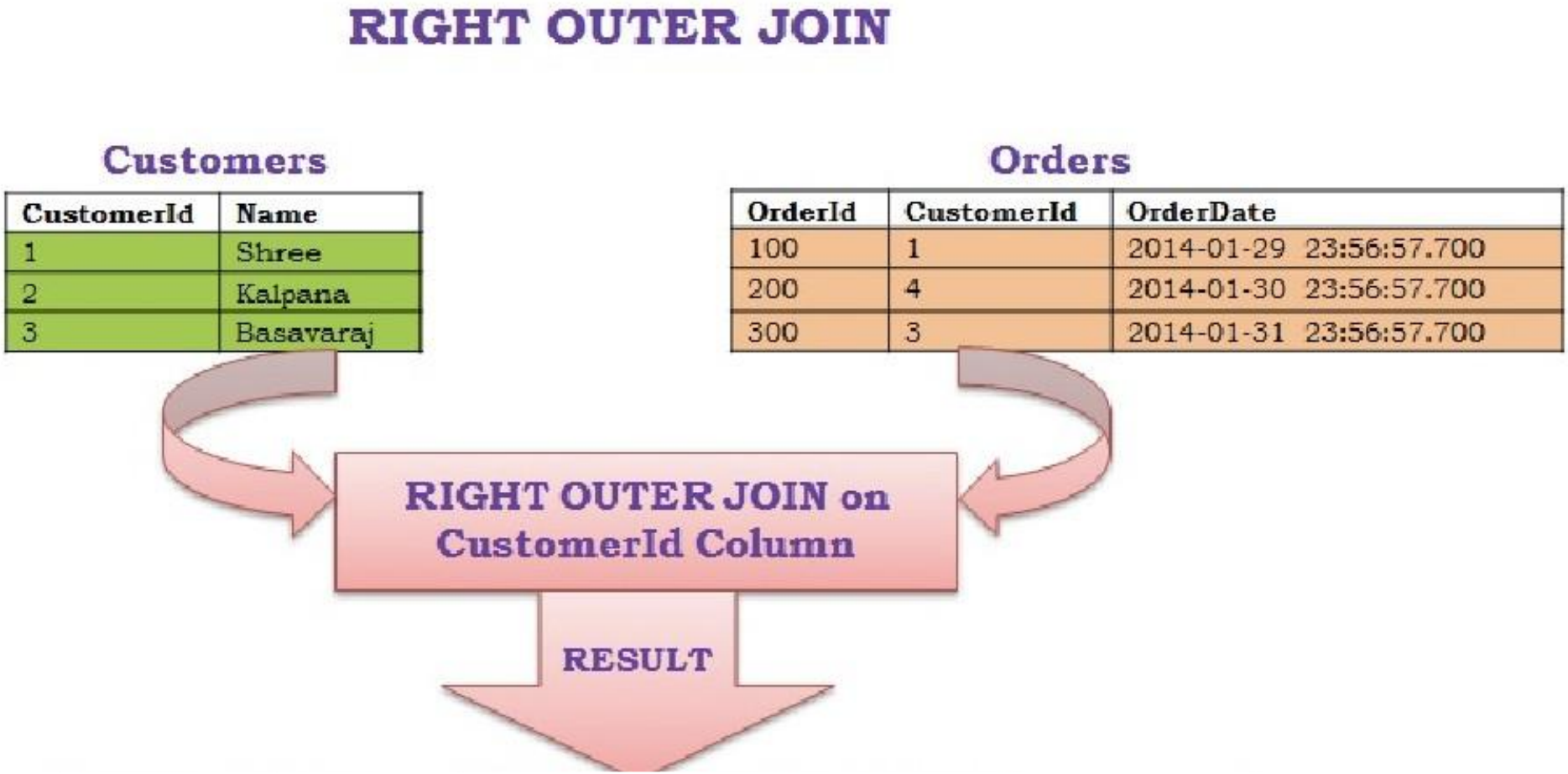
- Right join, sometimes called right outer join
- The RIGHT JOIN keyword returns all the rows from the right table (Category), even if there are no matches in the left table (Product).
- *A list of all categories and their matching products even if a category is not associated to a product.*

If no categories reference a particular product, that product does not show up

```
SELECT p.name, c.name
FROM Product p
RIGHT JOIN category c ON p.categoryID = c.categoryID
```

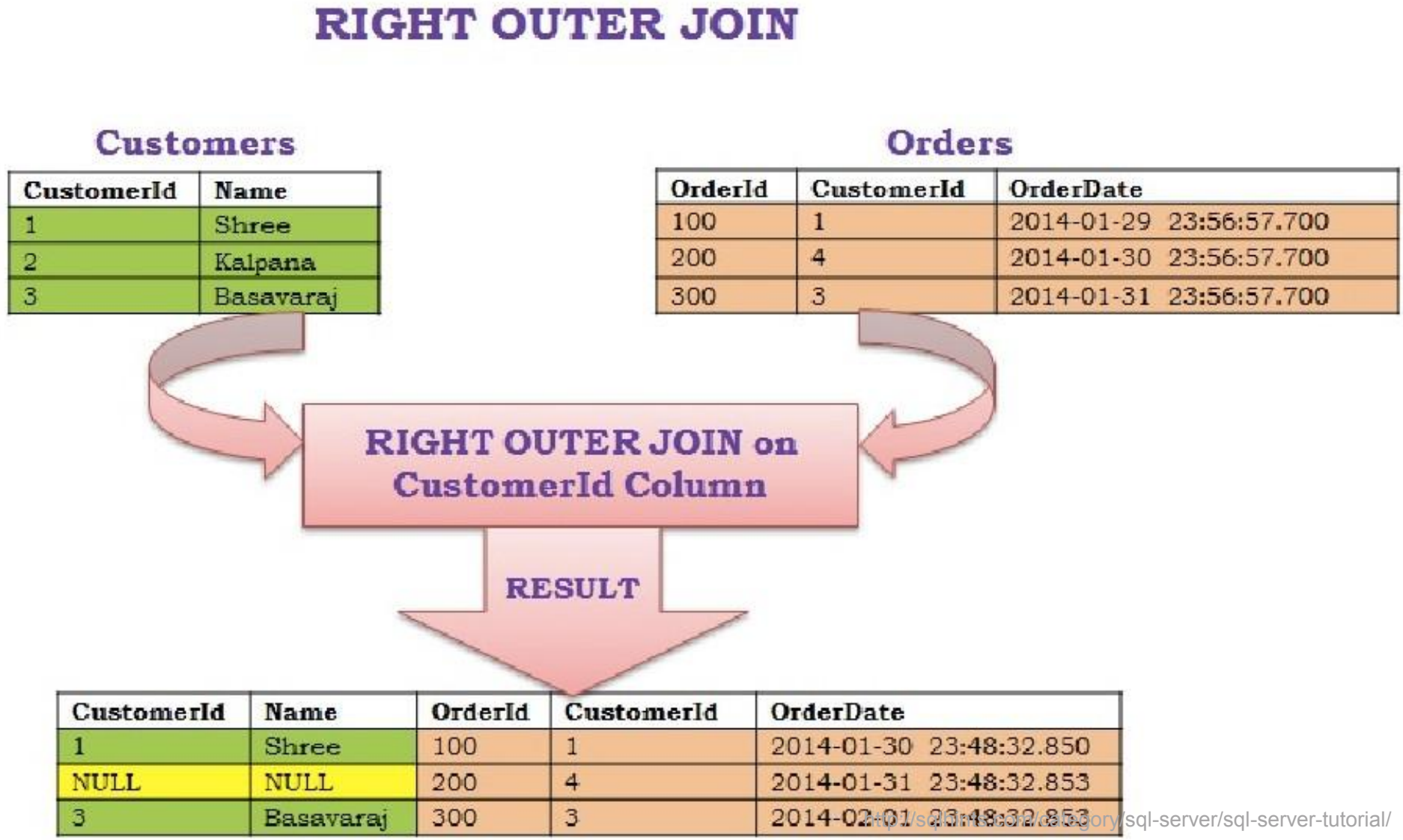
Right Outer Join

```
SELECT *
FROM  Customers C
        RIGHT OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```

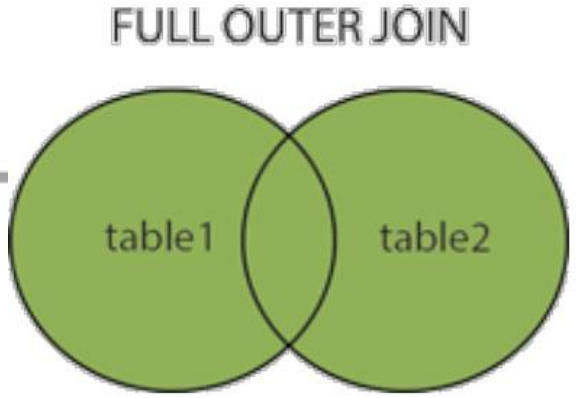


Right Outer Join

```
SELECT *
FROM  Customers C
        RIGHT OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```



Full Outer Join

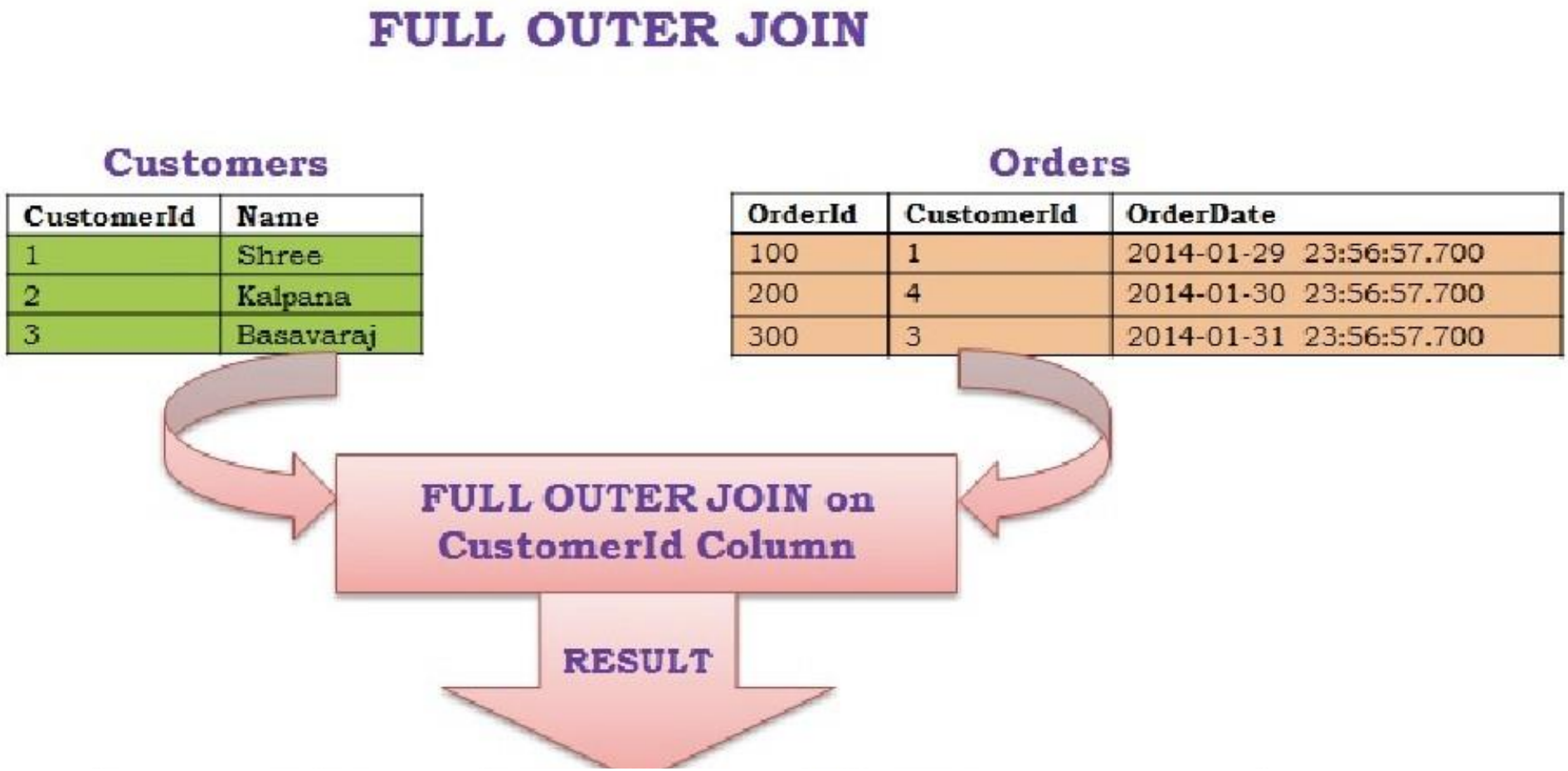


- Full outer join
- The FULL OUTER JOIN keyword returns all the rows from both tables.

```
SELECT p.name, c.name
FROM Product p
FULL OUTER JOIN category c ON p.categoryID = c.categoryID
```

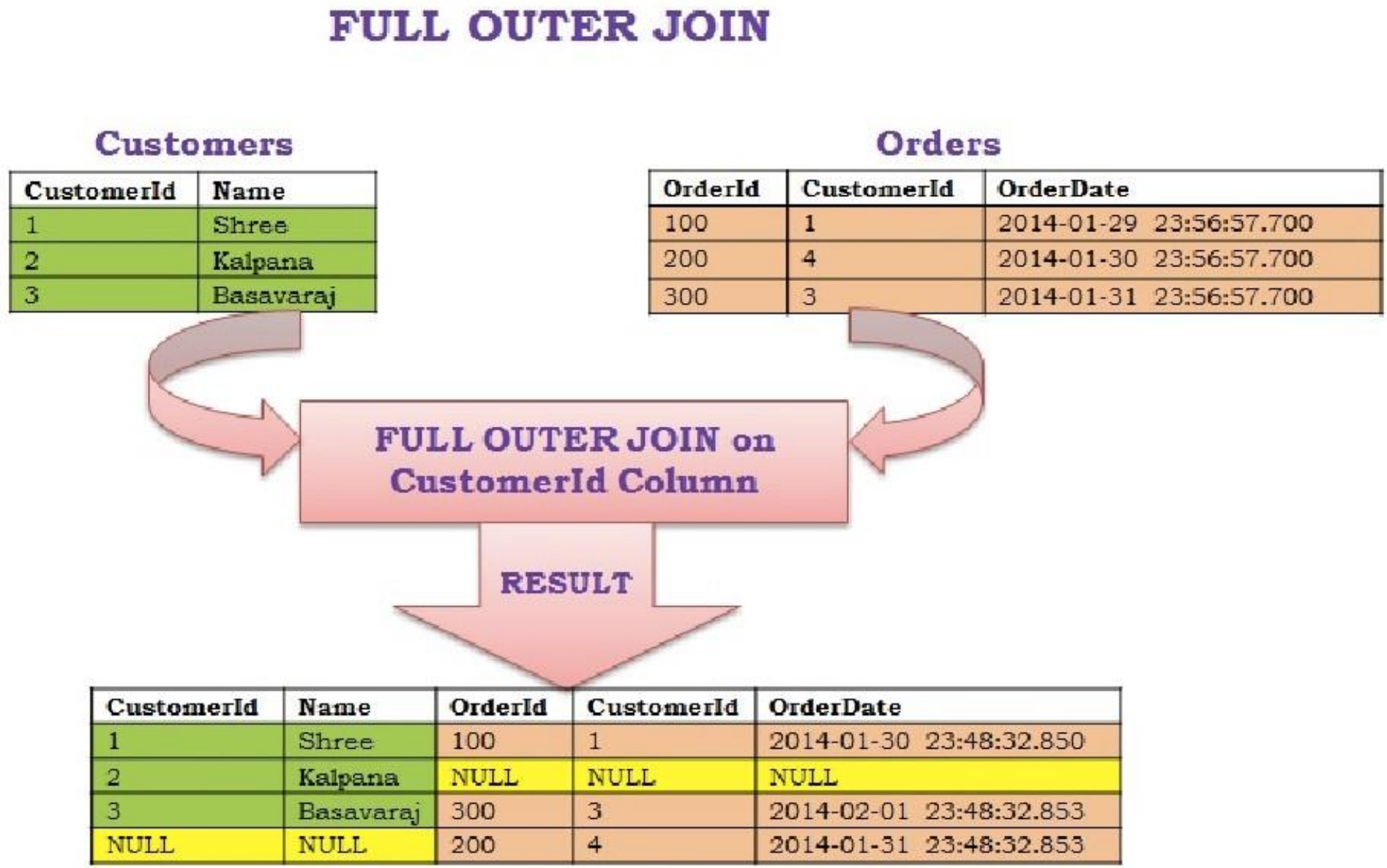
Full Outer Join

```
SELECT *
    FROM Customers C
        FULL OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```



Full Outer Join

```
SELECT *
    FROM Customers C
        FULL OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```



Categories

CategoryID	Category Name
10	Fruits
20	Vegetables

Products

ProductID	ProductName	Price	CategoryID
1	Apples	0.40	10
2	Oranges	1.10	10
3	Lettuce	0.60	20
4	Squash	1.20	20

How to get average price of products in each Category?

Aggregate Functions

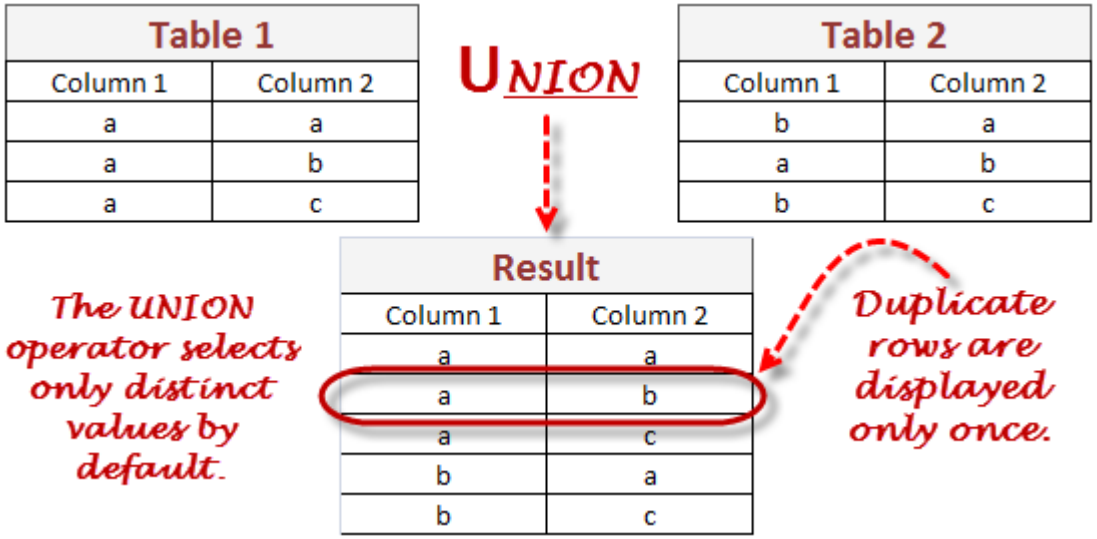
- Return a single value calculated from values in the column.
- Examples: **AVG(), COUNT(), MAX(), MIN(), SUM()**
- Can use a "**GROUP BY**" clause that includes all columns not aggregated to achieve results by groups

```
SELECT AVG(p.Price) 'Average Price'
,c.Name 'Category Name'
,c.Description 'Category Description'
FROM Product p JOIN Category c
ON p.categoryID = c.idCategory
GROUP BY c.Name, c.Desription
```

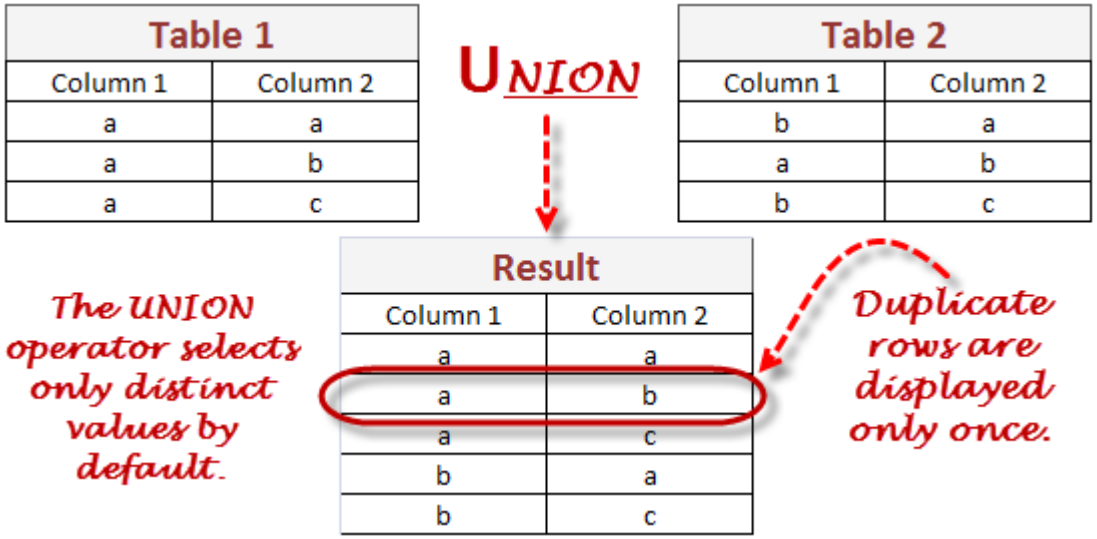
Unions

- Union operator is used to combine the result-set of two select statements.
  - Default is to remove duplicates
  - Include duplicates with: UNION ALL
- Column number and type must match

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```



What could be the query which resulted in the above result set from base tables – Table1 and Table2?




```
Select Column1, Column2 from Table1
UNION
Select Column1, Column2 from Table2;
```

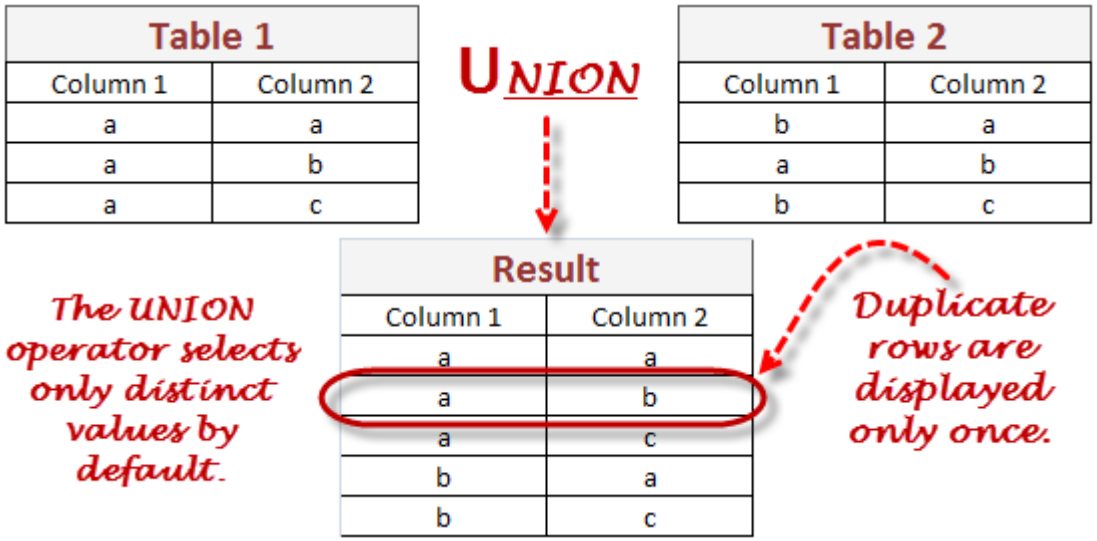


Table 1
Column 1
Column 2
a
a
a
b
a
c
UNION
Table 2
Column 1
Column 2
b
a
a
b
b
c
Result
Column 1
Column 2
a
a
a
b
a
c
b
a
b
c
The UNION operator selects only distinct values by default.
Duplicate rows are displayed only once.

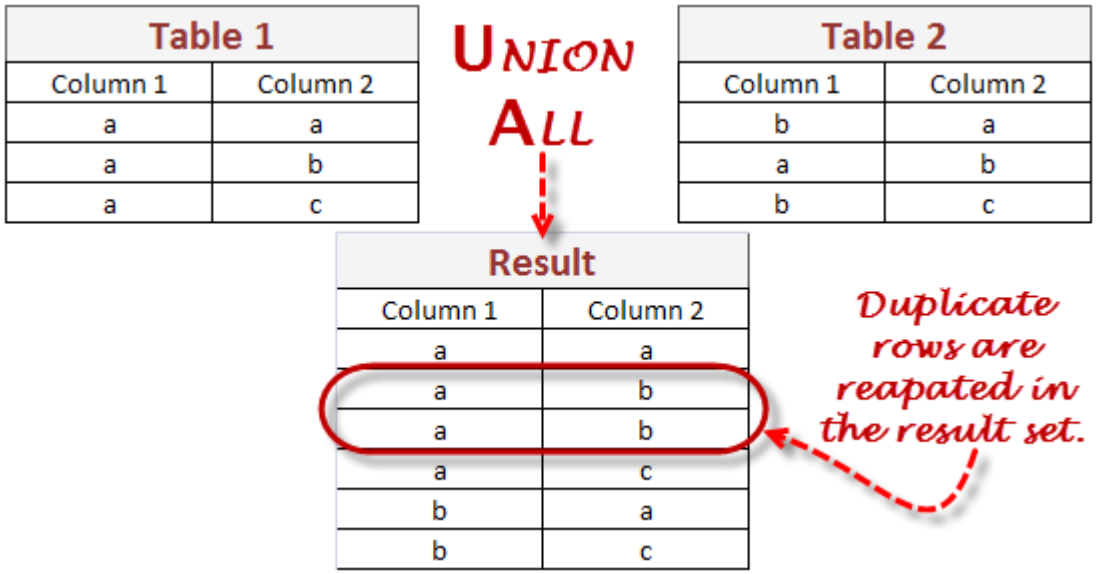


Table 1
Column 1
Column 2
a
a
a
b
a
c
UNION ALL
Table 2
Column 1
Column 2
b
a
a
b
b
c
Result
Column 1
Column 2
a
a
a
b
a
b
a
c
b
a
b
c
Duplicate rows are reapated in the result set.

Modifying Data

- Update statement used to change existing data.

```
UPDATE table_name
SET column1 = value1, column2 = value2,...
WHERE some_column = some_value;
```

- Delete used to remove records

```
DELETE FROM table_name
WHERE some_column = some_value;
```

- Insert used to create new records

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

For this Lab:

1. Check if you already have a mysql installation:
   sudo netstat -tap | grep mysql

2. If no, Install mysql:
   sudo apt-get install mysql-server

3. Connect to mysql using the password you configured during installation:
   sudo mysql -u root –p

4. Follow the Lab5 writeup, complete the database setup and write all the queries to retrieve or modify the tables in the database.

5. The last step of the writeup asks you to create an ER Diagram, for that we need to use some tool like MySQL Workbench. You are free to use any other tools.

- How to install mysql workbench:
  sudo apt-get install mysql-workbench

- After installing, connect to your database instance and select the database and try to **reverse engineer**, where the tool creates the ER Diagram based on the tables existing in your database.