Database Systems

Bilal Khalid Dar

Agenda

- How to retrieve data from database using SELECT

and:

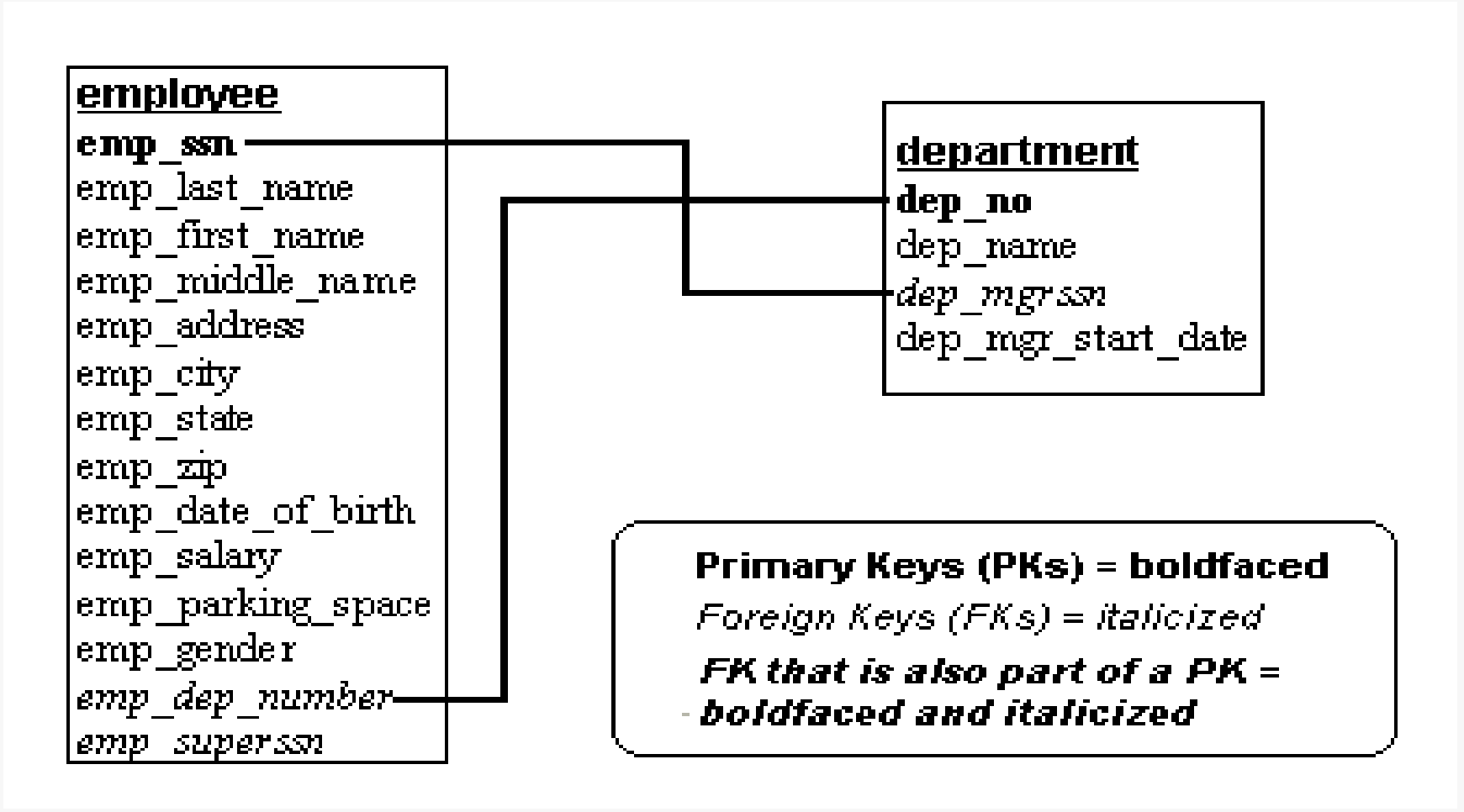
– Join tables together

- Set (Union, Intersection, Exist/Except)

A TYPICAL JOIN OPERATION

- Following figure displays the *employee* and *department* tables.
- The figure illustrates the concept of a JOIN operation by connecting columns within each table with a line.
- One line connects the *employee* table's *emp_dpt_number* column with the *department* table's *dpt_no* column.
- A second line connects the *employee* table's *emp_ssn* column to the *department* table's *dep_mgrssn* column.

A TYPICAL JOIN OPERATION



JOINS

- We will begin our study of JOIN operations by focusing on the relationship between the *employee* and *department* tables represented by the common department number values.
- Our first query lists employee names and department numbers. This query only retrieves data from the single *employee* table.

JOINS

```
SELECT emp_last_name "Last Name", emp_first_name "First Name",
    emp_dpt_number "Department"
FROM employee;
Last Name First Name  Department
------------ ------------- -------------
Bordoloi     Bijoy                    1
Joyner       Suzanne                   3
Zhu          Waiman                    7
more rows will be displayed . . .
```

JOINS

- A large organization can have dozens or even hundreds of departments. Thus, the numbers displayed in the department column shown above may not be very meaningful.
- Suppose you want the department names instead of the department numbers to be listed.
- The department names are mentioned in the “Department” table.
- Hence we need to join the “Employee” and the “Department” tables to get the required results

JOINS

```
SELECT emp_last_name "Last Name",
    emp_first_name "First Name",
    dpt_name "Department Name"
FROM employee, department
WHERE employee.emp_dpt_number = department.dpt_no;
Last Name  First Name Department Name
------------  ------------ ---------------------
Bordoloi      Bijoy        Headquarters
Joyner        Suzanne      Admin and Records
Zhu           Waiman       Production
```

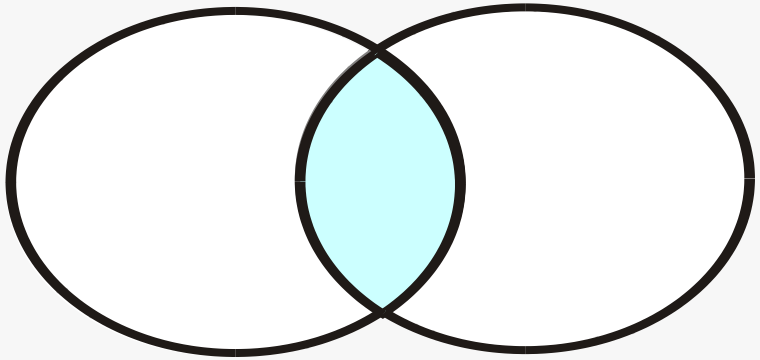
more rows will be displayed . . .

Types of Joins

- SQL supports two types of joins:
  - inner joins
  - outer joins

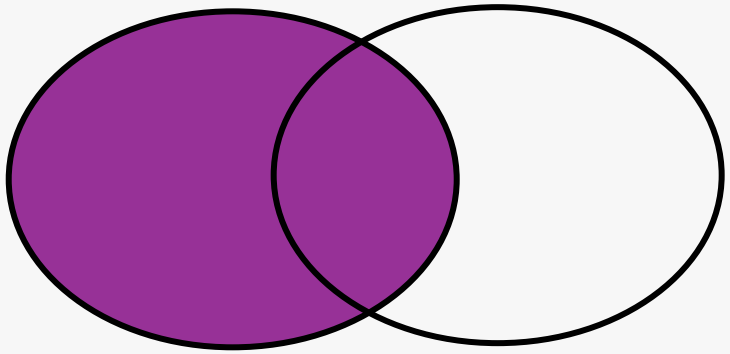
Types of Joins

- Inner joins
  - return only matching rows
  - enable a maximum of 256 tables to be joined at the same time.

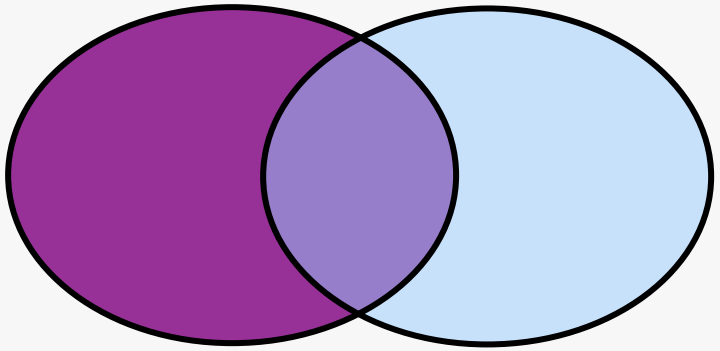


Types of Joins

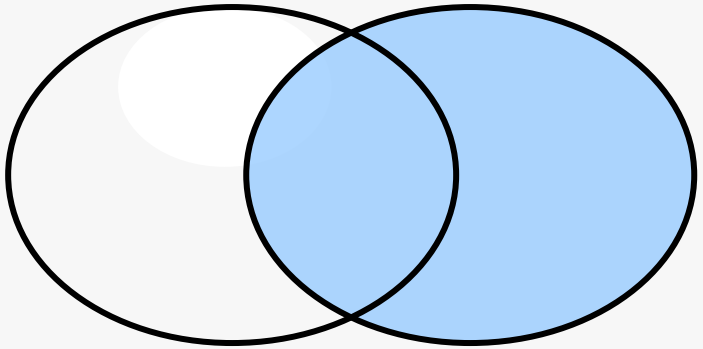
- Outer joins
  - return all matching rows, plus nonmatching rows from one or both tables
  - can be performed on only two tables or views at a time.



Left



Full



Right

Cartesian Product

- To understand how SQL processes a join, it is important to understand the concept of the Cartesian product.
- A query that lists multiple tables in the FROM clause without a WHERE clause produces all possible combinations of rows from all tables. This result is called the *Cartesian product*.

```
select *
   from one, two;
```

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y
4	d	5	v

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y
4	d	5	v
2	b	2	x

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y
4	d	5	v
2	b	2	x
2	b	3	y

Cartesian Product

Table One

X	A
1	a
4	d
2	b

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y
4	d	5	v
2	b	2	x
2	b	3	y
2	b	5	v

Cartesian Product

Table One

X	A
1	a
4	d
2	b

3 rows

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y
4	d	5	v
2	b	2	x
2	b	3	y
2	b	5	v

...

Cartesian Product

Table One

X	A
1	a
4	d
2	b

3 rows

3 rows

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y
4	d	5	v
2	b	2	x
2	b	3	y
2	b	5	v

...

Cartesian Product

Table One

X	A
1	a
4	d
2	b

3 rows

X

3 rows

Table Two

X	B
2	x
3	y
5	v

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y
4	d	5	v
2	b	2	x
2	b	3	y
2	b	5	v

9 rows

Cartesian Product

- The number of rows in a Cartesian product is the product of the number of rows in the contributing tables.

- 3 x 3 = 9

1,000 x 1,000 = 1,000,000

100,000 x 100,000 = 10,000,000,000

A Cartesian product is rarely the **desired** result of a query.

Poll

Quiz

5.02 Quiz

- How many rows are returned from this query?

```
select *
   from three, four;
```

Table Three

X	A
1	a1
1	a2
2	b1
2	b2
4	d

Table Four

X	B
2	x1
2	x2
3	y
5	v

5.02 Quiz – Correct Answer

The query produces 20 rows.

- How many rows are returned from this query?

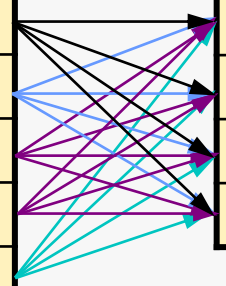
```
select *
   from three, four;
```

Table Three

X	A
1	a1
1	a2
2	b1
2	b2
4	d

Table Four

X	B
2	x1
2	x2
3	y
5	v



5*4=20

Partial Results Set

X	A	X	B
1	a1	2	x1
1	a1	2	x2
1	a1	3	y
1	a1	5	v
1	a2	2	x1
1	a2	2	x2
1	a2	3	y
1	a2	5	v
2	b1	2	x1
2	b1	2	x2

s105a01

JOINS

- The first row of the “table_1” table was joined with every row in the “table_2” table.
- A Cartesian product may not be useful and could be misleading.
- Always include a WHERE clause in your JOIN statements.

```
SELECT *
FROM table_1, table_2;
WHERE table_1.col_1 = table_2.col_1;
```

col_1	col_1
a	a
b	b
c	c

JOIN OPERATION RULES

JOINS and the SELECT Clause

- A JOIN query always begins with a SELECT clause.
- List the columns to be displayed in the result table after the SELECT keyword.
- The result table column order reflects the order in which column names are listed in the SELECT clause.
- To modify the order in which column names are listed, simply rearrange the order of the column listing in the SELECT clause.

Example

```
SELECT dpt_name "Department Name",
    emp_last_name "Last Name",
    emp_first_name "First Name"
FROM employee e, department d
WHERE e.emp_dpt_number = d.dpt_no AND e.emp_dpt_number = 7;
```

```
Department Name Last Name    First Name
--------------------- ------------- ---------------
Production           Zhu           Waiman
Production           Bock          Douglas
Production           Joshi         Dinesh
Production           Prescott       Sherri
```

JOINS and the FROM Clause

- Any SELECT statement that has two or more table names listed in a FROM clause is a JOIN query.
- By definition, a JOIN operation retrieves rows from two or more tables.
- Always the FROM clause is used to list the tables from which columns are to be retrieved by a JOIN query.
- The FROM clause listing has a limit of 16 table names.
- The order of table name listings is irrelevant to the production of the result table with the one exception – that is, if you use an asterisk (*) in the SELECT clause, then the column order in the result table reflects the order in which tables are listed in the FROM clause.

JOINS and the WHERE Clause

- The WHERE clause specifies the relationship between tables listed in the FROM clause.
- It also restricts the rows displayed in the result table.
- The most commonly used JOIN operator is the "equal" (=) sign.

QUALIFYING COLUMN NAMES

- When column names are ambiguous (the column names used are from more than one table) you must qualify them

```
SELECT *
FROM table_1, table_2
WHERE col_1 = col_1;
```

- This query is **WRONG**, Oracle Server would reject this query and generate the error message "ambiguous object.“

```
Error at line 3:
ORA - 00918 : column ambiguously defined
```

QUALIFYING COLUMN NAMES

- This error message tells you that you have included a column name somewhere in the query that exists in more than one table listed in the FROM clause.
- Here the error is in the WHERE clause; however, it is also possible to make a similar error in the SELECT clause.
- The SELECT statement shown below fails to qualify the col_1 name in the SELECT clause, and Oracle again produces the ORA-00918 error message.

```
SELECT col_1
FROM table_1, table_2
WHERE table_1.col_1 = table_2.col_1;
ERROR at line 1:
ORA-00918: column ambiguously defined
```

QUALIFYING COLUMN NAMES

- An ambiguous column name is qualified by using the DOT (.) connector to connect the table name and column name.
- Sometimes it is easier to qualify column names by using table alias names.
- Often, a single letter is used as an identifier to reduce keystroke requirements .

```
SELECT dpt_name "Department Name",
   emp_last_name "Last Name",
   emp_first_name "First Name"
FROM employee e, department d
WHERE e.emp_dpt_number = d.dpt_no AND e.emp_dpt_number = 7;
```

QUALIFYING COLUMN NAMES

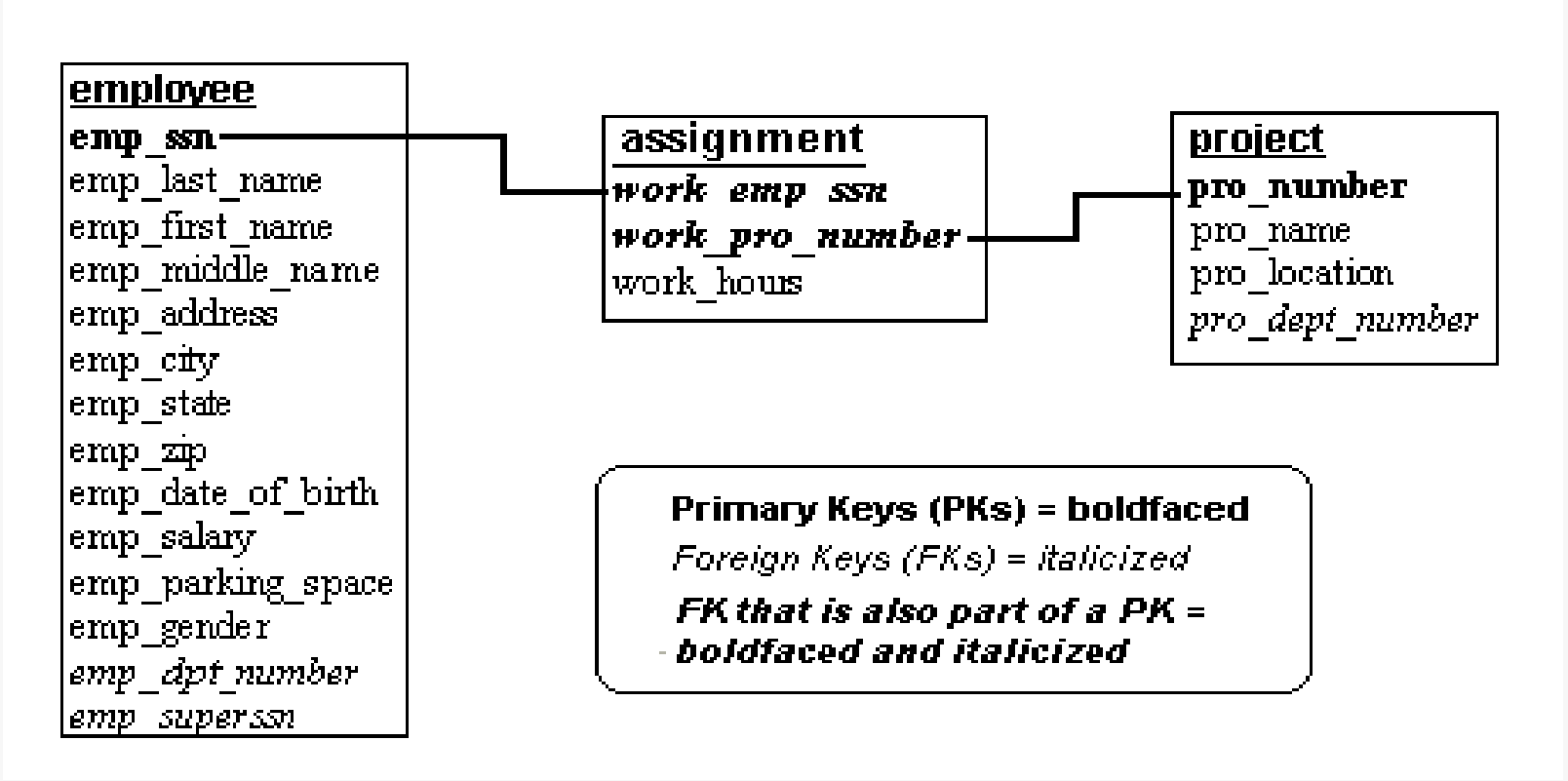
- The use of the letters "e" and "d" is completely arbitrary; "t1" and "t2" or any other unique aliases could have been used.
- The important points to learn are:
  - The alias must follow a table name.
  - Use a space to separate a table name and its alias.
  - The alias must be unique within the SELECT statement.
- If the column names are not identical you are not required to qualify them, although you still might want to for documentation purposes

Joining More Than Two Tables

- While the examples given thus far have joined rows from two tables, you can specify up to 16 tables in a JOIN operation.
- The more tables that are included in a JOIN operation, the longer the query will take to process, especially when the tables are large with millions of rows per table.

Joining More Than Two Tables

- The example shown in following figure joins three tables to produce a result table based on two different relationships.



Joining More Than Two Tables

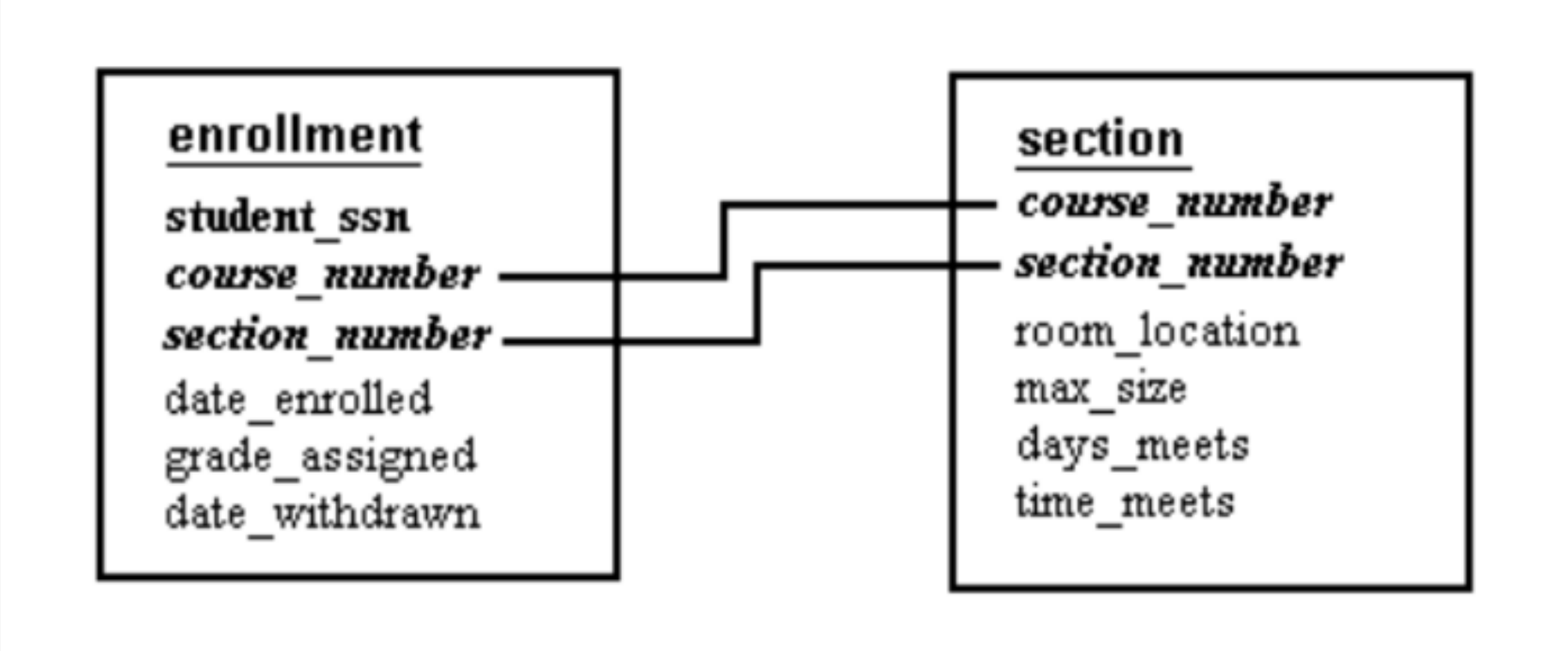
- The SELECT statement to join the tables depicted in the figure is shown here.

```
SELECT emp_last_name "Last Name",
    emp_first_name "First Name",
    1.10*emp_salary "Raised Salary", p.pro_name "Project"
FROM employee e, assignment a, project p
WHERE e.emp_ssn = a.work_emp_ssn AND
    a.work_pro_number = p.pro_number AND
    p.pro_name = 'Inventory';
```

```
Last Name    First Name    Raised Salary Project
-------------- -------------- ---------------- -----------
Zhu            Waiman          $47,300    Inventory
Markis          Marcia          $27,500    Inventory
Amin           Hyder           $27,500    Inventory
```

Joining Tables by Using Two Columns

- The diagram depicts the relationship at a university where students enroll in course sections.



Joining Tables by Using Two Columns

- The SELECT statement that accomplishes the JOIN based on two columns is shown below.
- This situation arises when the related tables have *composite primary key* columns.

```
SELECT s.course_title "Course Title",
    e.student_ssn "Student SSN"
FROM enrollment e, section s
WHERE e.course_number = s.course_number AND
   e.section_number = s.section_number;
```

BREAK

OUTER JOIN Operations

- Oracle also supports what is called an outer-join. This means that a row will appear in the joined table even though there is no matching value in the table to be joined.
- Suppose you want to know the names of the employees regardless of whether they have dependents or not. You can use the outer join as follows

OUTER JOIN Operations

- The plus sign in parentheses (+) tells Oracle to execute an OUTER JOIN operation.
- Further, it is the *dependent* table that is being outer-joined to the *employee* table because some employees will not have dependents.

```
SELECT emp_last_name "Last Name", emp_first_name "First Name",
   dep_name "Dependent",
   dep_relationship "Relationship"
FROM employee e, dependent d
WHERE e.emp_ssn = d.dep_emp_ssn(+);
```

```
SELECT emp_last_name "Last Name",
 emp_first_name "First Name",
 dep_name "Dependent",
 dep_relationship "Relationship"
FROM employee e, dependent d
WHERE e.emp_ssn = d.dep_emp_ssn(+);

Last Name       First Name      Dependent      Relationship
--------------- --------------- -------------- ------------
Bordoloi        Bijoy
Joyner          Suzanne         Allen          SPOUSE
Zhu             Waiman          Andrew         SON
Zhu             Waiman          Jo Ellen       DAUGHTER
Zhu             Waiman          Susan          SPOUSE
Markis          Marcia
Amin            Hyder
Bock            Douglas         Deanna         DAUGHTER
Bock            Douglas         Jeffery        SON
Bock            Douglas         Mary Ellen     SPOUSE
Joshi           Dinesh
Prescott        Sherri
12 rows selected.
```

OUTER JOINS and NULL values

- Management might desire a listing of employees with no dependents in order to satisfy some governmental reporting requirement.
- We can take advantage of the fact that the *dep_name* column will be NULL for employees with no dependents, and simply add a criteria to the WHERE clause to include employees where the *dep_name* column is NULL.

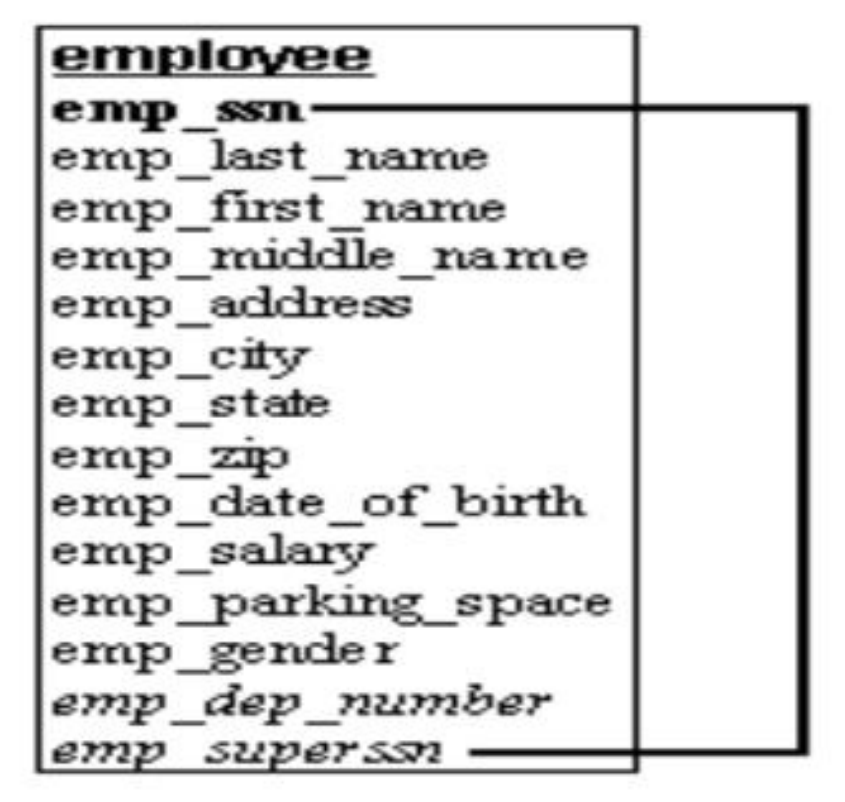
OUTER JOINS and NULL values

```
SELECT emp_last_name "Last Name", emp_first_name "First Name"
FROM employee e, dependent d
WHERE e.emp_ssn = d.dep_emp_ssn(+) AND
   d.dep_name IS NULL;
```

Last Name	First Name
Bordoloi	Bijoy
Markis	Marcia
Amin	Hyder
Joshi	Dinesh
Prescott	Sherri

SELF-JOIN Operations

- A SELF JOIN operation is used to produce a result table when the relationship of interest exists among rows that are stored within a single table.



SELF-JOIN Operations

```
SELECT e1.emp_last_name || ', ' || e1.emp_first_name "Supervisor",
   e2.emp_last_name || ', ' || e2.emp_first_name "Employee"
FROM employee e1, employee e2
WHERE e1.emp_ssn = e2.emp_superssn;
```

Supervisor	Employee
Bordoloi, Bijoy	Joyner, Suzanne
Bordoloi, Bijoy	Zhu, Waiman
Joyner, Suzanne	Markis, Marcia
Joyner, Suzanne	Amin, Hyder
Zhu, Waiman	Bock, Douglas
Zhu, Waiman	Joshi, Dinesh
Zhu, Waiman	Prescott, Sherri

Alternative JOIN Constructs

- **SQL provides alternative ways to specify joins:**

  FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo

  FROM Client JOIN Viewing USING clientNo

  FROM Client NATURAL JOIN Viewing

- **FROM replaces original FROM and WHERE**

Example Sorting a join

For each branch, list numbers and names of staff who manage properties, and properties they manage.

```
SELECT s.branchNo, s.staffNo, fName, lName,
                propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, propertyNo;
```

Example 6.25 Sorting a join

Table 5.25 Result table for Example 5.25.

branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

Example Three Table Join

For each branch, list staff who manage properties, including city in which branch is located and properties they manage.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName,
                propertyNo
FROM Branch b, Staff s, PropertyForRent p
WHERE b.branchNo = s.branchNo AND
                s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

Example 6.26 Three Table Join

Table 5.26 Result table for Example 5.26.

branchNo	city	staffNo	fName	lName	propertyNo
B003	Glasgow	SG14	David	Ford	PG16
B003	Glasgow	SG37	Ann	Beech	PG21
B003	Glasgow	SG37	Ann	Beech	PG36
B005	London	SL41	Julie	Lee	PL94
B007	Aberdeen	SA9	Mary	Howe	PA14

Example Multiple Grouping Columns

Find number of properties handled by each staff member by branch.

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS myCount
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

Example 6.27 Multiple Grouping Columns

branchNo	staffNo	myCount
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Computing a Join

Procedure for generating results of a join are:

1. **Form Cartesian product of tables named in FROM clause**
2. **If WHERE clause:**
   - **Apply search condition to each row of product table**
   - **Retain rows that satisfy condition**
3. **For each remaining row, determine value of each item in SELECT list to produce single row in result table**

Computing a Join

4. If DISTINCT specified, eliminate any duplicate rows from result table

6. If ORDER BY clause, sort result table as required

Outer Joins

- If one row of joined table is unmatched, row omitted from result table
- Outer join operations retain rows that do not satisfy join condition
- Consider following tables:

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Outer Joins

- **The (inner) join of these two tables:**

```
SELECT b.*, p.*
FROM Branch1 b, PropertyForRent1 p
WHERE b.bCity = p.pCity;
```

Table 5.27(b) Result table for inner join of Branch1 and PropertyForRent1 tables.

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Example Left Outer Join

List branches and properties that are in same city along with any unmatched branches.

```
SELECT b.*, p.*
FROM Branch1 b LEFT JOIN
    PropertyForRent1 p ON b.bCity = p.pCity;
```

Example Left Outer Join

- **Includes rows of first (left) table unmatched with rows from second (right) table**
- **Columns from second table filled with NULLs**

Table 5.28 Result table for Example 5.28.

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

Example 6.29 Right Outer Join

List branches and properties in same city and any unmatched properties.

```
SELECT b.*, p.*
FROM Branch1 b RIGHT JOIN
  PropertyForRent1 p ON b.bCity = p.pCity;
```

Example 6.29 Right Outer Join

- **Right Outer join includes rows of second (right) table unmatched with rows from first (left) table**
- **Columns from first table filled with NULLs**

Table 5.29 Result table for Example 5.29.

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Example Full Outer Join

List branches and properties in same city and any unmatched branches or properties.

```
SELECT b.*, p.*
FROM Branch1 b FULL JOIN
  PropertyForRent1 p ON b.bCity = p.pCity;
```

Example Full Outer Join

- **Includes rows unmatched in both tables**
- **Unmatched columns filled with NULLs**

Table 5.30 Result table for Example 5.30.

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS used only with subqueries
- Produce simple true/false result
- True if and only if there exists at least one row in result table returned by subquery
- False if subquery returns empty result table
- NOT EXISTS is the opposite of EXISTS

EXISTS and NOT EXISTS

- As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns
- Common for subqueries following (NOT) EXISTS to be of form:

  (SELECT * ...)

Example Query using EXISTS

Find all staff who work in a London branch.

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS
   (SELECT *
    FROM Branch b
    WHERE s.branchNo = b.branchNo AND
                city = 'London');
```

Example Query using EXISTS

Table 5.31 Result table for Example 5.31.

staffNo	fName	lName	position
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

Example Query using EXISTS

- Note, search condition s.branchNo = b.branchNo is necessary to consider correct branch record for each member of staff
- If omitted, would get all staff records listed out because subquery:

```
SELECT * FROM Branch WHERE city='London'
```

- would always be true and query would be:

```
SELECT staffNo, fName, lName, position FROM Staff
WHERE true;
```

Example Query using EXISTS

- **Could also write this query using join construct:**

```
SELECT staffNo, fName, lName, position
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
                   city = 'London';
```

Union, Intersect, and Difference (Except)

- Can use normal set operations of Union, Intersection, and Difference to combine results of two or more queries into single result table
- Union of two tables, A and B, is table containing all rows in either A or B or both
- Intersection is table containing all rows common to both A and B
- Difference is table containing all rows in A but not in B
- Two tables must be *union compatible*

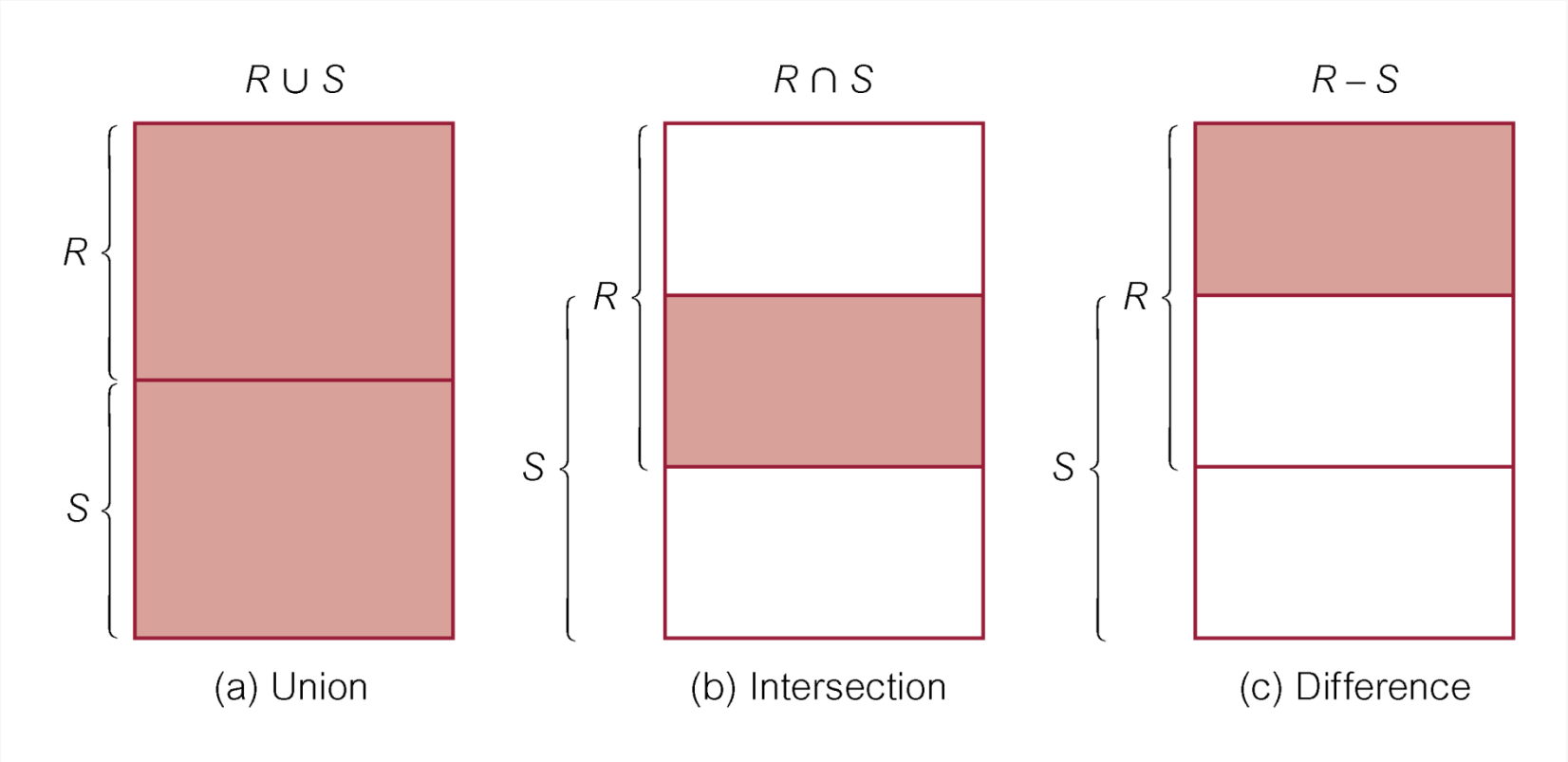
Union, Intersect, and Difference (Except)

- **Format of set operator clause in each case is:**

  op **[ALL] [CORRESPONDING [BY {column1 [, ...]}]]**

- **If CORRESPONDING BY specified, set operation performed on the named column(s)**
- **If CORRESPONDING specified but not BY clause, operation performed on common columns**
- **If ALL specified, result can include duplicate rows**

Union, Intersect, and Difference (Except)



Example Use of UNION

List all cities where there is either a branch office or a property.

```
(SELECT city
FROM Branch
WHERE city IS NOT NULL) UNION
(SELECT city
FROM PropertyForRent
WHERE city IS NOT NULL);
```

Example Use of UNION

- **Produces result tables from both queries and merges both tables together.**

Table 5.32 Result table for Example 5.32.

city
London
Glasgow
Aberdeen
Bristol

Example Use of INTERSECT

List all cities where there is both a branch office and a property.

```
(SELECT city FROM Branch)
INTERSECT
(SELECT city FROM PropertyForRent);
```

Example Use of INTERSECT

- **Or**

```
(SELECT * FROM Branch)
INTERSECT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

Table 5.33 Result table for Example 5.33.

city
Aberdeen
Glasgow
London

Example Use of INTERSECT

- Could rewrite this query without INTERSECT operator:

```
SELECT b.city
FROM Branch b PropertyForRent p
WHERE b.city = p.city;
```

- Or:

```
SELECT DISTINCT city FROM Branch b
 WHERE EXISTS
   (SELECT * FROM PropertyForRent p
   WHERE p.city = b.city);
```

Example Use of EXCEPT

List of all cities where there is a branch office but no properties.

```
(SELECT city FROM Branch)
EXCEPT
(SELECT city FROM PropertyForRent);
```

- Or

```
(SELECT * FROM Branch)
EXCEPT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

Table 5.34 Result table for Example 5.34.

city
Bristol

Example Use of EXCEPT

- **Could rewrite this query without EXCEPT:**

```
SELECT DISTINCT city FROM Branch
WHERE city NOT IN
  (SELECT city FROM PropertyForRent);
```

- **Or**

```
SELECT DISTINCT city FROM Branch b
WHERE NOT EXISTS
  (SELECT * FROM PropertyForRent p
  WHERE p.city = b.city);
```

Sample Quiz

Customer Table

- CREATE TABLE Customers (
- CustomerID INT PRIMARY KEY,
- CustomerName VARCHAR(50),
- Country VARCHAR(50)
-);

Order Table

- CREATE TABLE Orders (
- OrderID INT PRIMARY KEY,
- CustomerID INT,
- OrderDate DATE
-);

Q1

- Find the names of customers who placed orders.

```
SELECT DISTINCT CustomerName
FROM Customers
INNER JOIN Orders ON
Customers.CustomerID =
Orders.CustomerID;
```

Q2

- List the order IDs and order dates for orders placed by customers from the USA.

```
SELECT Orders.OrderID, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID
= Customers.CustomerID
WHERE Customers.Country = 'USA';
```

Q3

- Retrieve the order details (OrderID, OrderDate) along with the customer name for each order.

```
SELECT Orders.OrderID, Orders.OrderDate,
Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID =
Customers.CustomerID;
```

Q4

- Show the customer names and their respective countries for customers who placed orders in February 2023.

```
SELECT DISTINCT Customers.CustomerName,
Customers.Country
FROM Customers
INNER JOIN Orders ON Customers.CustomerID =
Orders.CustomerID
WHERE Orders.OrderDate BETWEEN '2023-02-01' AND
'2023-02-28';
```

Q5

- Find the total number of orders placed by each customer.

```
SELECT Customers.CustomerName,
COUNT(Orders.OrderID) AS TotalOrders
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID
GROUP BY Customers.CustomerName;
```

Q6

- Display the customer names who didn't place any orders.

```
SELECT Customers.CustomerName
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID
WHERE Orders.OrderID IS NULL;
```


Thank
You