Database Management Systems:
Fundamentals and Introduction to SQL

SQL Queries

Three Categories of SQL Commands

Data Control Language (DCL)

- Commands used to control a database, including those for administering privileges and committing (saving) data

Data Definition Language (DDL)

- Commands used to define a database, including those for creating, altering, and dropping tables and establishing constraints
- Statements that return no result set
- As a general rule, statements of this type generally change the database in some way
- CREATE, ALTER, DROP

Three Categories of SQL Commands

Data Manipulation Language (DML)

- Commands used to maintain and query a database, including those for updating, inserting, modifying, and querying data
- Return a result set
- INSERT, UPDATE, DELETE, and SELECT

DDL vs DML

- Change the information or the DB itself
- Retrieve information from the DB

DDL

SQL Database Definition

- Data Definition Language (DDL)
- CREATE, ALTER, DROP

- Major CREATE statements:
  - **CREATE SCHEMA – defines a portion of the database owned by a particular user**
  - **CREATE TABLE – defines a new table and its columns**
  - CREATE VIEW – defines a logical table from one or more tables or views

Changing Table Definitions

- Syntax:

ALTER TABLE table_name *alter_table_actions;*

- ALTER TABLE statement allows you to change column specifications

Alter_table_actions examples:

ADD [COLUMN] col_name column_definition
ALTER [COLUMN] col_name SET DEFAULT default-value | DROP DEFAULT
DROP [COLUMN] col_name
DROP PRIMARY KEY
DROP FOREIGN KEY fk_symbol

CREATE TABLE with AUTO_INCREMENT

- The AUTO_INCREMENT attribute can be used to generate a unique identity for new rows
- AUTO_INCREMENT columns cannot contain NULL values
- AUTO_INCREMENT columns must be indexed. i.e. UNIQUE, PRIMARY KEY

AUTO_INCREMENT Example

```
CREATE TABLE if not exists animals (
    id          INT          NOT NULL AUTO_INCREMENT,
    name        VARCHAR(30)  NOT NULL,
    PRIMARY KEY (id)
);

INSERT INTO animals (name)
VALUES ('dog'),('cat'),('penguin'),('lax'),('whale'),('ostrich');
```

- AUTO_INCREMENT VALUES can be reset,
  - INSERT INTO animals (id,name) VALUES(100,'rabbit');
  - INSERT INTO animals (id,name) VALUES(NULL,'mouse');
  - ALTER TABLE animals AUTO_INCREMENT = 10;

DML

SELECT Statement

- Used for queries on single or multiple tables

- **Selecting ALL data**
  - **use * specifier**
- **Select particular COLUMNS**
- **Select particular ROWS**
  - **WHERE clause**
- Sorting rows
  - ORDER BY
- Counting rows

SELECT Statement Syntax

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

- what_to_select indicates what you want to see. This can be a list of columns, or * to indicate “all columns.”
- which_table indicates the table from which you want to retrieve data.
- The WHERE clause is optional. If it is present, conditions_to_satisfy specifies one or more conditions that rows must satisfy to qualify for retrieval.

SELECT with Conditions: WHERE

- Conditions – WHERE clause
  - equality
  - inequality
  - pattern matches

Table 12.4 Comparison Operators

Name	Description
`>`	Greater than operator
`>=`	Greater than or equal operator
`<`	Less than operator
`<>, !=`	Not equal operator
`<=`	Less than or equal operator
`<=>`	NULL-safe equal to operator
`=`	Equal operator
`BETWEEN ... AND ...`	Whether a value is within a range of values
`COALESCE()`	Return the first non-NULL argument
`GREATEST()`	Return the largest argument
`IN()`	Whether a value is within a set of values

MySQL 8.0 Reference Manual

SELECT with Conditions

Comparison operators (cont')

Name	Description
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

SELECT with Conditions: Pattern Matches

- **LIKE** operator
  - a logical operator that tests whether a string contains a specified pattern or not
- Two wildcard characters for constructing patterns
  - The percentage (%) wildcard matches <u>any string of zero, one, or more characters</u>
  - The underscore (_) wildcard matches <u>one, single character</u>

- e.g.,
  - **'s%'** matches any string starting with the character **s**, such as so, sql, see, sea, seed
  - 'se_' matches any string starting with se and is followed by another character, such as **see** and **sea** but not **seed**

SELECT with Multiple Conditions

- WHERE clause can be used for multiple conditions
- Boolean Operators

AND	Joins two or more conditions and returns results only when all conditions are true.
OR	Joins two or more conditions and returns results when any conditions are true.
NOT	Negates an expression.

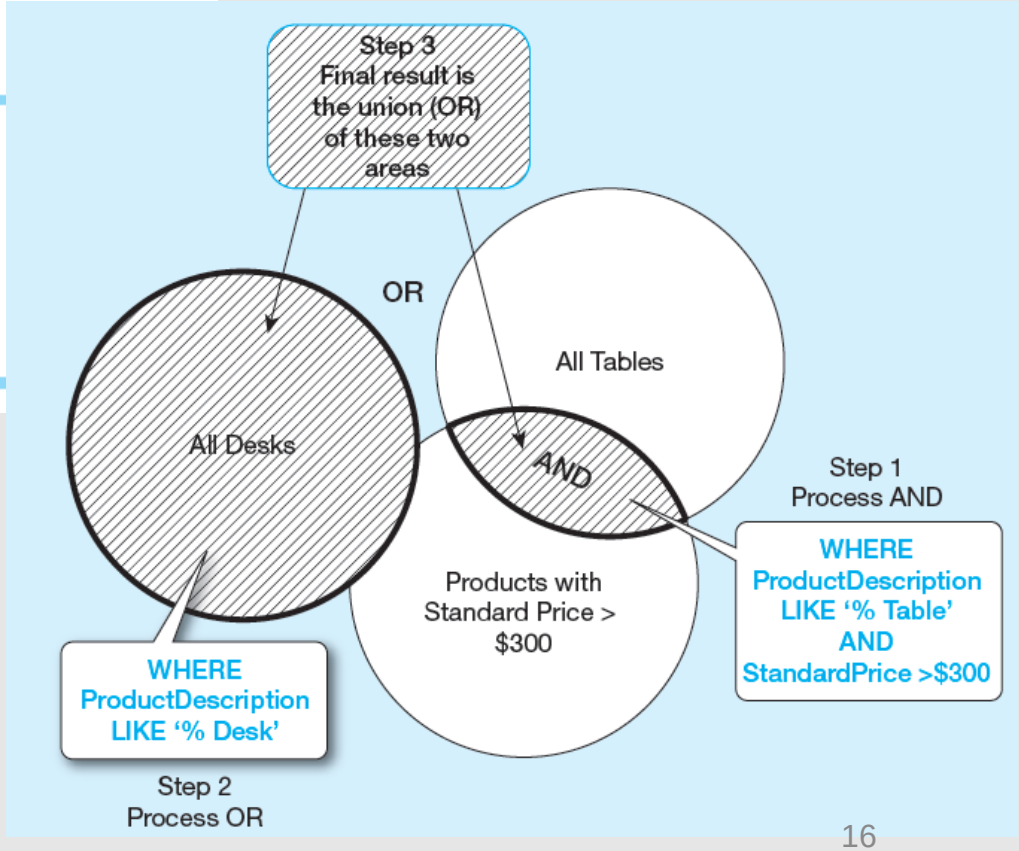
>= AND <= = BETWEEN … AND …

Using Boolean Operators

Operator Precedence: **OR** < **AND**

Query A: List product name, finish, and standard price for all desks and all tables that cost more than $300 in the Product table.

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
  FROM Product_T
    WHERE ProductDescription LIKE '%Desk'
      OR ProductDescription LIKE '%Table'
      AND ProductStandardPrice > 300;
```

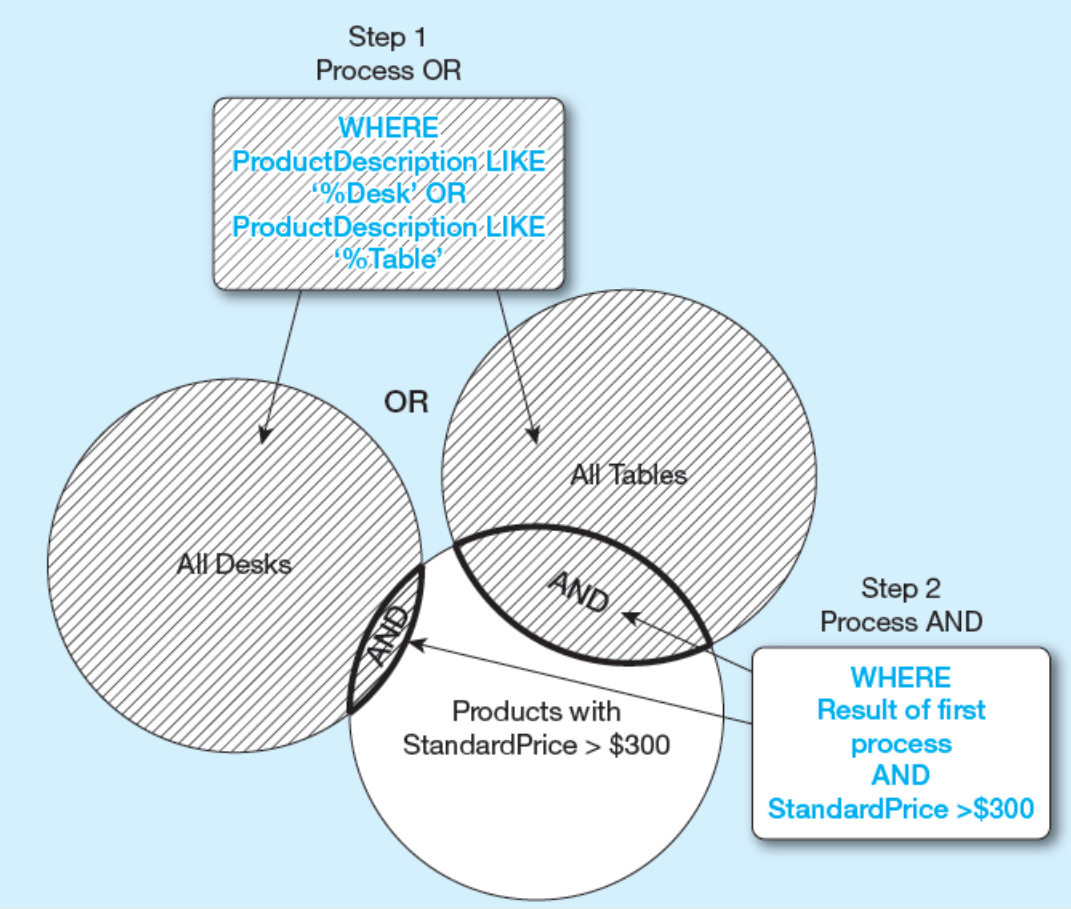


- the results include All Desks regardless of the price
- To override the order and group terms explicitly, **use parentheses**.

Using Boolean Operators

Operator Precedence: **( )** > **AND**

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
   FROM Product_T;
   WHERE (ProductDescription LIKE '%Desk'
      OR ProductDescription LIKE '%Table')
      AND ProductStandardPrice > 300;
```



SELECT with Conditions: IN, NOT IN

Using IN and NOT IN with Lists

Query: List all customers who live in warmer states.

```
SELECT CustomerName, CustomerCity, CustomerState
  FROM Customer_T
    WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI');
```

Result:

CUSTOMERNAME	CUSTOMERCITY	CUSTOMERSTATE
Contemporary Casuals	Gainesville	FL
Value Furniture	Plano	TX
Impressions	Sacramento	CA
California Classics	Santa Clara	CA
M and H Casual Furniture	Clearwater	FL
Seminole Interiors	Seminole	FL
Kaneohe Homes	Kaneohe	HI
7 rows selected.		

Note: The IN operator in this example allows you to include rows whose CustomerState value is either FL, TX, CA, or HI.
It is more efficient than separate OR conditions.

Creating New Columns with SELECT Statement

- Output columns with expressions or calculations
  - SELECT col1, col2, CONCAT(col3, ’characters’, col4)
    FROM tbl_name;

character to connect two columns

- Using aliases for new column
  - *concise, readability*

  - SELECT col1, col2, CONCAT(col3, ’characters’, col4) AS ‘new_col_name’
    FROM tbl_name;
  - e.g., SELECT t, size/1024 AS kilobytes FROM mail

SELECT Distinct Values

- Get the distinct values
- remove duplicates

- SELECT DISTINCT col_name FROM tbl_name;

Work with Null Values

- A null value means that a column is missing a value; the value is not zero or blank or any special code—there simply is no value
- It is not uncommon to first explore whether there are null values before deciding how to write other commands
- simply need to see data rows where there are missing values
- NULL values behave differently when used by sorting and summary operations

- IS NULL, IS NOT NULL

SELECT Statement

- Selecting ALL data
  - use * specifier
- Select particular COLUMNS
- Select particular ROWS
  - WHERE clause
- **Counting rows**
  - **COUNT()**
- **Sorting rows**
  - **ORDER BY**

Count Rows

- A type of summary operations
- SELECT **COUNT**(*) FROM …
- COUNT (*) vs COUNT(colname)
  - COUNT (*) counts all rows selected by a query, regardless of whether any of the rows contain null values
  - COUNT(colname) tallies only rows that contain values; it ignores all null values

Sorting Query Results

- MySQL returns rows in any order
- Bring order to disorder to make query results easier to examine and understand

SELECT… ORDER BY…

- Sorting using one column
- Sorting using multiple columns
- Sorting in ascending order (the default) or descending order
  - specified after each column
  - can be mixed-order sorting
- Sorting using expressions and aliases
  - WHERE conditions only work on original column names
  - ORDER BY can use alias

Sorting Query Results

- sort values defined as ‘strings’ instead of ‘numbers’

```
CREATE TABLE roster
(
  name        CHAR(30),   # player name
  jersey_num  CHAR(3),     # jersey number
  PRIMARY KEY(name)
);
```

```
+-----------+------------+
| name      | jersey_num |
+-----------+------------+
| Lynne     | 29         |
| Ella      | 0          |
| Elizabeth | 100        |
| Nancy     | 00         |
| Jean      | 8          |
| Sherry    | 47         |
+-----------+------------+
```

```
SELECT name, jersey_num FROM roster ORDER BY jersey_num;
```

```
+-----------+------------+
| name      | jersey_num |
+-----------+------------+
| Ella      | 0          |
| Nancy     | 00         |
| Elizabeth | 100        |
| Lynne     | 29         |
| Sherry    | 47         |
| Jean      | 8          |
+-----------+------------+
```

```
SELECT name, jersey_num FROM roster ORDER BY jersey_num+0;
```

```
+-----------+------------+
| name      | jersey_num |
+-----------+------------+
| Ella      | 0          |
| Nancy     | 00         |
| Jean      | 8          |
| Lynne     | 29         |
| Sherry    | 47         |
| Elizabeth | 100        |
+-----------+------------+
```

string-to-number conversion

Sorting Query Results

- Sorting using multiple columns : the sequence of attributes after 'ORDER BY' matters

```
SELECT last_name, first_name FROM name
ORDER BY last_name, first_name;
```

LIMIT Clause for Query Result

- Retrieve an arbitrary section of a result set
- Split the result into sections

- Select.. LIMIT n;
  - return *at most* n rows

LIMIT Clause for Query Result

- two-argument form of LIMIT : …… LIMIT n,m;
  - n rows to skip
  - m rows to return
  - e.g. Q: What is the third-smallest value? (min(), max() not suitable)
  - n=? m=?

- partition a result set into smaller sections
  - RETURN 20 rows at a time
  - SELECT …… LIMIT a , b;
  - SELECT …… LIMIT c , d;
  - SELECT …… LIMIT e , f;
  - a,b,c,d,e,f, = ?

DELETE Statement

- Removes rows from a table
- Delete certain rows
  - DELETE FROM Customer_T WHERE CustomerState = ‘HI’;
- Delete all rows
  - DELETE FROM Customer_T;

UPDATE Statement

- Modifies data in existing rows
  - UPDATE Product_T
    SET ProductStandardPrice = 775
    WHERE ProductID = 7;

Complex SQL queries

Work with Multiple Tables

- obtain more comprehensive information than obtained from individual tables
- hold intermediate-result for a multi-stage operation
- modify data in one table based on information from another

Create New Table using Existing Ones

- Save Query Result in a Table
  - Safe to work with a copy
  - speed up the process
  - preliminary checks and corrections by generating a temporary table
  - perform summary operations on a large table more efficiently, avoid running repeatedly

  - CREATE TABLE … SELECT…;

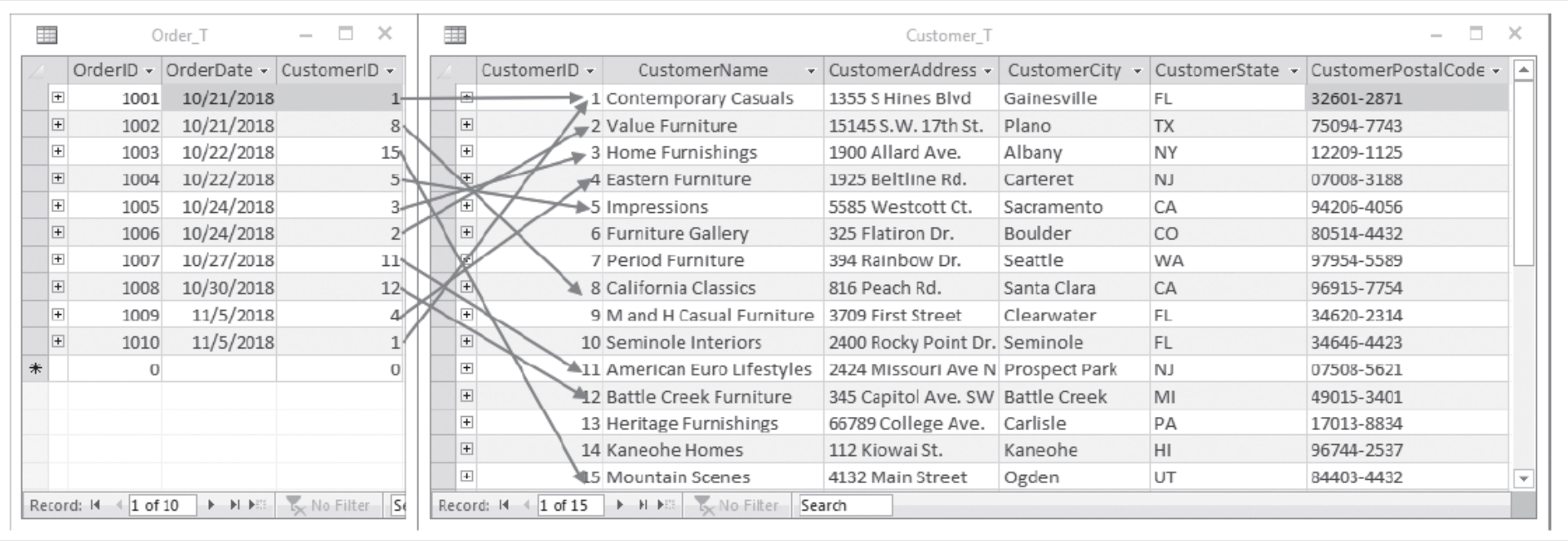
Work with Multiple Tables

- LIMITATIONs of **CREATE TABLE …. SELECT**
  - **PRIMARY KEY, AUTO_INCREMENT**, columns' **default value** are not copied to the destination table, need to specify them explicitly
- Using cloning table technique to create an exact copy

- Cloning a Table
  - Create a table that has exactly the same structure as an existing table
  - CREATE TABLE new_tbl LIKE original_tbl;

Finding Matches Between Tables

- **Join**
- A relational operation that causes two tables with a common domain to be combined into a single table or view
- Use primary - foreign key relationship



Order_T

OrderID	OrderDate	CustomerID
1001	10/21/2018	1
1002	10/21/2018	8
1003	10/22/2018	15
1004	10/22/2018	5
1005	10/24/2018	3
1006	10/24/2018	2
1007	10/27/2018	11
1008	10/30/2018	12
1009	11/5/2018	4
1010	11/5/2018	1
0		0

Record: 1 of 10 No Filter

Customer_T

CustomerID	CustomerName	CustomerAddress	CustomerCity	CustomerState	CustomerPostalCode
1	Contemporary Casuals	1355 S Hines Blvd	Gainesville	FL	32601-2871
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094-7743
3	Home Furnishings	1900 Allard Ave.	Albany	NY	12209-1125
4	Eastern Furniture	1925 Beltline Rd.	Carteret	NJ	07008-3188
5	Impressions	5585 Westcott Ct.	Sacramento	CA	94206-4056
6	Furniture Gallery	325 Flatiron Dr.	Boulder	CO	80514-4432
7	Period Furniture	394 Rainbow Dr.	Seattle	WA	97954-5589
8	California Classics	816 Peach Rd.	Santa Clara	CA	96915-7754
9	M and H Casual Furniture	3709 First Street	Clearwater	FL	34620-2314
10	Seminole Interiors	2400 Rocky Point Dr.	Seminole	FL	34646-4423
11	American Euro Lifestyles	2424 Missouri Ave N	Prospect Park	NJ	07508-5621
12	Battle Creek Furniture	345 Capitol Ave. SW	Battle Creek	MI	49015-3401
13	Heritage Furnishings	66789 College Ave.	Carlisle	PA	17013-8834
14	Kaneohe Homes	112 Kiowai St.	Kaneohe	HI	96744-2537
15	Mountain Scenes	4132 Main Street	Ogden	UT	84403-4432

Record: 1 of 15 No Filter Search

Finding Matches Between Tables

- **Join**
- A complete join that produces all possible row combinations is called a *Cartesian product*
- A join of a 200-row table and a 100-row table =>200*100 = 20000 rows

Finding Matches Between Tables

- *artist.sql*
- SELECT * FROM artist **JOIN** painting ORDER BY artist.a_id;

a_id	name	a_id	p_id	title	state	price
1	Da Vinci	1	1	The Last Supper	IN	34
1	Da Vinci	3	3	Starry Night	KY	48
1	Da Vinci	4	5	Les Deux Soeurs	NE	64
1	Da Vinci	1	2	Mona Lisa	MI	87
1	Da Vinci	3	4	The Potato Eaters	KY	67
2	Monet	1	2	Mona Lisa	MI	87
2	Monet	3	4	The Potato Eaters	KY	67
2	Monet	1	1	The Last Supper	IN	34
2	Monet	3	3	Starry Night	KY	48
2	Monet	4	5	Les Deux Soeurs	NE	64
3	Van Gogh	1	2	Mona Lisa	MI	87
3	Van Gogh	3	4	The Potato Eaters	KY	67
3	Van Gogh	1	1	The Last Supper	IN	34
3	Van Gogh	3	3	Starry Night	KY	48
3	Van Gogh	4	5	Les Deux Soeurs	NE	64
4	Renoir	1	1	The Last Supper	IN	34
4	Renoir	3	3	Starry Night	KY	48
4	Renoir	4	5	Les Deux Soeurs	NE	64
4	Renoir	1	2	Mona Lisa	MI	87
4	Renoir	3	4	The Potato Eaters	KY	67

no restrictions on row matching

JOIN Types

- Inner join (Equi-join)
  - A join in which the joining condition is based on equality between values in the common columns.
  - Common columns appear (redundantly) in the result table
- Outer join
  - A join in which rows that do not have matching values in common columns are nevertheless included in the result table

Inner Join

- SELECT … FROM *table1* INNER JOIN *table2 conditions*
- Join conditions
  - WHERE
  - ON
  - USING
    - same names in tables
    - the column appears once in result

- ON/USING – how to join tables
- WHERE – which of the joined rows to select

Joining Tables from Different Databases

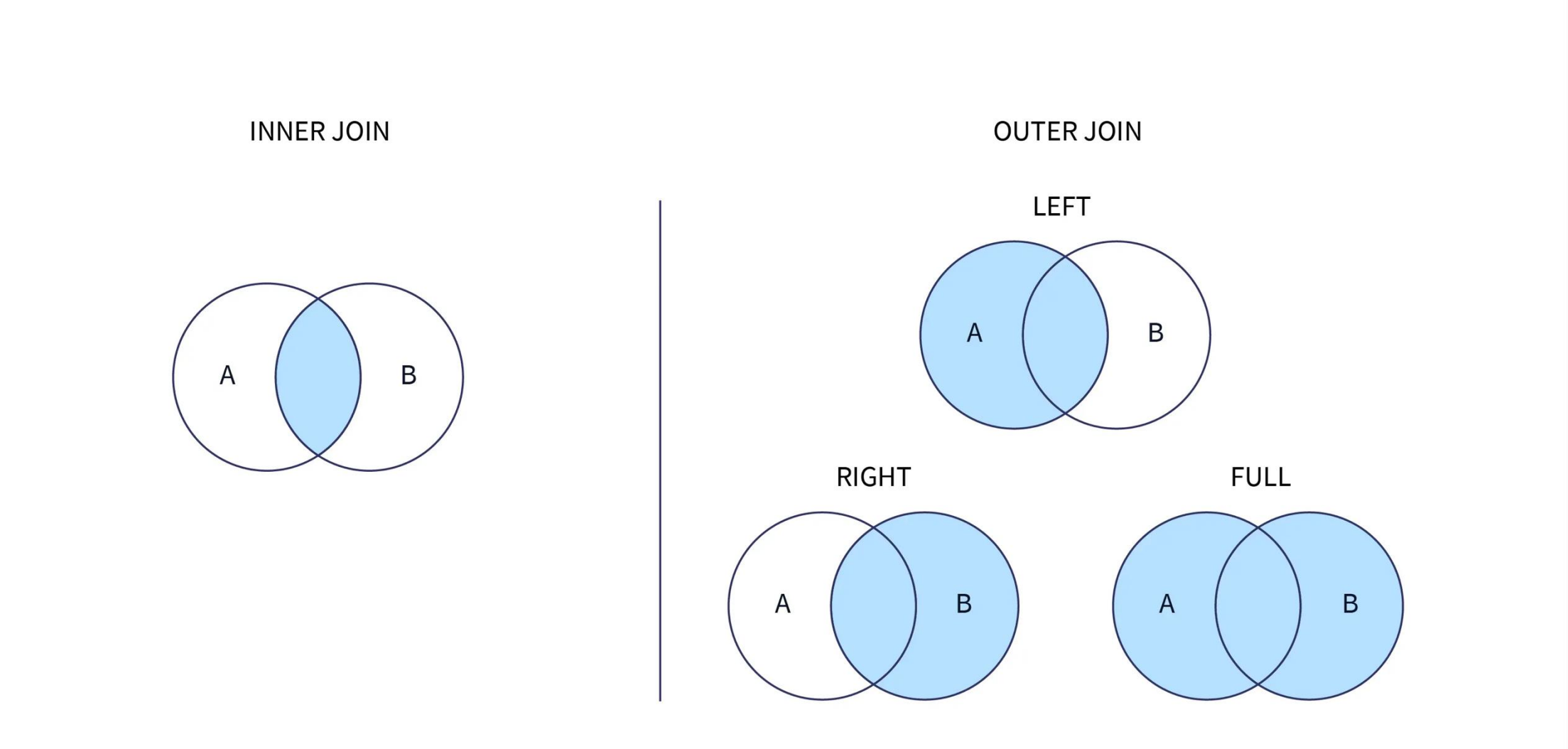
- SELECT db1.artist.name, db2.painting.title
  FROM db1.artist INNER JOIN db2.painting
  ON db1.artist.a_id = db2.painting.a_id;

- SELECT a.name, p.title ### a,p - Alias for DB.table
  FROM db1.artist AS a INNER JOIN db2.painting AS p
  ON a.a_id = p.a_id;

Outer Join

- Joins written with INNER JOIN produce a result only for values in one table that **match** values in another table.
- An outer join can produce those matches as well but also can show you which values in one table are **missing** from the other.
- e.g., *You have a list of potential customers and another list of people who have placed orders. To focus sales efforts on people who are not yet actual customers, produce the set of people who are in the first list but not the second.*

Outer Join vs Inner Join



Outer Join

- *Q: Which artists in the artist table are missing from the painting table?*

```
mysql> SELECT * FROM artist INNER JOIN painting
    -> ON artist.a_id <> painting.a_id
    -> ORDER BY artist.a_id;
+------+----------+------+------+-------------------+-------+-------+
| a_id | name     | a_id | p_id | title             | state | price |
+------+----------+------+------+-------------------+-------+-------+
|    1 | Da Vinci |    4 |    5 | Les Deux Soeurs   | NE    |    64 |
|    1 | Da Vinci |    3 |    4 | The Potato Eaters | KY    |    67 |
|    1 | Da Vinci |    3 |    3 | Starry Night      | KY    |    48 |
|    2 | Monet    |    1 |    1 | The Last Supper   | IN    |    34 |
|    2 | Monet    |    4 |    5 | Les Deux Soeurs   | NE    |    64 |
|    2 | Monet    |    3 |    4 | The Potato Eaters | KY    |    67 |
|    2 | Monet    |    3 |    3 | Starry Night      | KY    |    48 |
|    2 | Monet    |    1 |    2 | Mona Lisa         | MI    |    87 |
|    3 | Van Gogh |    1 |    2 | Mona Lisa         | MI    |    87 |
|    3 | Van Gogh |    1 |    1 | The Last Supper   | IN    |    34 |
|    3 | Van Gogh |    4 |    5 | Les Deux Soeurs   | NE    |    64 |
|    4 | Renoir   |    3 |    3 | Starry Night      | KY    |    48 |
|    4 | Renoir   |    1 |    2 | Mona Lisa         | MI    |    87 |
|    4 | Renoir   |    1 |    1 | The Last Supper   | IN    |    34 |
|    4 | Renoir   |    3 |    4 | The Potato Eaters | KY    |    67 |
+------+----------+------+------+-------------------+-------+-------+
```

INNER JOIN only produce results based on values that are present in both tables

Outer Join

- LEFT JOIN
  - match rows in the left table with rows in the second table
  - if no match, a row with all the columns from the right table are set to NULL

```
mysql> SELECT * FROM artist INNER JOIN painting
    -> ON artist.a_id = painting.a_id
    -> ORDER BY artist.a_id;
+------+----------+------+------+------------------+-------+-------+
| a_id | name     | a_id | p_id | title            | state | price |
+------+----------+------+------+------------------+-------+-------+
|    1 | Da Vinci |    1 |    1 | The Last Supper  | IN    |    34 |
|    1 | Da Vinci |    1 |    2 | Mona Lisa        | MI    |    87 |
|    3 | Van Gogh |    3 |    3 | Starry Night     | KY    |    48 |
|    3 | Van Gogh |    3 |    4 | The Potato Eaters | KY    |    67 |
|    4 | Renoir   |    4 |    5 | Les Deux Soeurs  | NE    |    64 |
+------+----------+------+------+------------------+-------+-------+
```

```
mysql> SELECT * FROM artist LEFT JOIN painting
    -> ON artist.a_id = painting.a_id
    -> ORDER BY artist.a_id;
+------+----------+------+------+------------------+-------+-------
| a_id | name     | a_id | p_id | title            | state | price
+------+----------+------+------+------------------+-------+-------
|    1 | Da Vinci |    1 |    1 | The Last Supper  | IN    |    34
|    1 | Da Vinci |    1 |    2 | Mona Lisa        | MI    |    87
|    2 | Monet    | NULL | NULL | NULL             | NULL  |  NULL
|    3 | Van Gogh |    3 |    3 | Starry Night     | KY    |    48
|    3 | Van Gogh |    3 |    4 | The Potato Eaters | KY    |    67
|    4 | Renoir   |    4 |    5 | Les Deux Soeurs  | NE    |    64
+------+----------+------+------+------------------+-------+-------
```

To finally get the exact output: **WHERE painting.a_id IS NULL**.

Outer Join

```
mysql> SELECT artist.* FROM artist LEFT JOIN painting
    -> ON artist.a_id = painting.a_id
    -> WHERE painting.a_id IS NULL;
+------+-------+
| a_id | name  |
+------+-------+
|    2 | Monet |
+------+-------+
```

USING NOT IN

```
mysql> SELECT * FROM artist
    -> WHERE a_id NOT IN (SELECT a_id FROM painting);

+------+-------+
| a_id | name  |
+------+-------+
|    2 | Monet |
+------+-------+
```

Outer Join

- Right Join
  - reversed case of the LEFT JOIN

  - table1 LEFT JOIN table2
  - =
  - table2 RIGHT JOIN table1

In-class Practice – Submit on Camino

- upload MySQL statements text of answers to the following questions
- *Which paintings did Van Gogh paint?*

```
+-------------------+
| title             |
+-------------------+
| Starry Night      |
| The Potato Eaters |
+-------------------+
```

- *Who painted the Mona Lisa?*

```
+----------+
| name     |
+----------+
| Da Vinci |
+----------+
```

- *For which artists did you purchase paintings in Kentucky or Indiana?*

```
+----------+
| name     |
+----------+
| Da Vinci |
| Van Gogh |
+----------+
```

What is next…

- Assignment 2
- Next week topics: joins, complex queries