

Where/ Distinct

SELECT * → DISTINCT → Top () * → From → Join → Where → Group By → Having → Order By

The screenshot displays the Microsoft SQL Server Management Studio (SSMS) interface. On the left, the Object Explorer shows the database structure for 'noble_desktop'. The 'Tables' folder is expanded, and 'dbo.line_items' and 'dbo.orders' are highlighted. The 'Columns' folder for 'dbo.orders' is also expanded, showing columns like 'order_id', 'user_id', 'created_at', 'ship_name', 'ship_address', 'ship_city', 'ship_state', and 'ship_zipcode'. The main SQL Editor window contains a series of SQL queries with comments explaining their purpose. The bottom pane shows the results of the last query, which is a table with two rows: 'NM' and 'VT'.

SQL Editor

```
-- 1. Find all rows from the orders table with user_id 30.
select * from orders where user_id =30

-- 2. Select all columns from line_items where someone ordered a quantity of 3 or more.
select * from line_items where quantity >=3

-- 3. Select the rows from line_items with a price less than $30.
select * from line_items where price<=30.00

-- 4. Select the rows from line_items with a price of $30 or more, ordered by most expensive first.
select * from line_items where price >=30 order by price desc

-- 5. Limit the results to just see the top 20 most expensive line_items.
select top(20)* from line_items order by price desc

-- 6. Find the orders that were shipped to zipcode 10499 or 77719
-- Keep in mind that zipcodes are stored as strings, not numbers!
select * from orders where ship_zipcode in ('10499', '77719')

-- 7. Modify the last query to see only the DISTINCT names of the people those orders were shipped to
select DISTINCT ship_name from orders

-- EXTRA CREDIT: If you finish early.

-- 1. View the 3 most recent orders made by user_id 33.
select top(3)* from orders where user_id = 33 order by user_id desc

-- 2. Use DISTINCT to find out which states user_id 33 has shipped orders to.
Select distinct ship_state from orders where user_id =33
```

Results

ship_state
1 NM
2 VT

View the Results

Query executed successfully. KAREY\SQLEXPRESS (15.0 RTM) KAREY\wkare (53) noble_desktop 00:00:00 2 rows

Wildcards/ Like

SELECT * → DISTINCT → Top () * → From → Join → Where → Group By → Having → Order By

The screenshot displays the Microsoft SQL Server Management Studio interface. The title bar indicates the file is '2.0 LIKE and Wildcards.sql' in the 'KAREY\SQLSERVER.noble_desktop (KAREY\wkare (63))' database. The menu bar includes File, Edit, View, Query, Project, Tools, Window, and Help. The toolbar contains icons for file operations, query execution, and formatting. The Object Explorer on the left shows the database structure, including 'KAREY\SQLSERVER (SQL Server 15.0.2000 - KAREY\wkare)', 'Databases', 'System Databases', 'Database Snapshots', 'noble_desktop', 'Security', 'Server Objects', 'Replication', 'PolyBase', 'Management', and 'XEvent Profiler'. A red box labeled 'Object Explorer' points to this pane. The SQL Editor in the center contains a query with six numbered comments and their corresponding SQL statements. A red box labeled 'SQL Editor' points to this pane. The query is as follows:

```
-- 1. Find all the users with a gmail email address.
select * from users where email like '%gmail%'

-- 2. Find all the orders shipped to Florida or Texas.
-- Bonus: Order the results by the state.
select * from orders where ship_state = 'fl' or ship_state = 'tx' order by ship_state
SELECT * FROM orders WHERE ship_state IN ('FL', 'TX');

-- 3. Find the 5 most recent orders shipped to New York.
select top (5) * from orders where ship_state = 'ny' order by ship_state desc

-- 4. Select all the products that include the word 'plate' and cost more than $20.
select * from products where title like '%plate%' and price > 20.00

-- 5. Find all the products that do NOT contain 'rubber' in the title.
select * from products where not title like '%rubber%'

-- 6. Find all the products that are tagged 'grey' or 'gray'
-- (notice the different spellings: one is 'e' and other 'a')
select * from products where tags = 'grey' or tags = 'gray'

-----
-- EXTRA CREDIT: If you finish early.
-----

-- 1. Find the most expensive items from line_items which status is 'returned'
select * from line_items where status = 'returned'

-- 2. You can perform math in ORDER BY.
-- ORDER BY price multiplied by quantity to find the most expensive overall returns.
select * from line_items where status = 'returned' order by quantity desc
```

The Results pane at the bottom shows the output of the last query, displaying 18 rows of data. A red box labeled 'View the Results' points to this pane. The data is as follows:

	line_item_id	order_id	product_id	price	quantity	status
1	51	38	5	63.22	10	returned
2	134	93	46	71.41	10	returned
3	97	69	33	60.24	10	returned
4	269	195	32	49.97	10	returned
5	275	198	3	48.64	10	returned
6	314	223	20	15.66	10	returned
7	436	298	50	88.19	10	returned
8	687	471	36	47.01	10	returned
9	786	38	5	63.22	10	returned
10	869	93	46	71.41	10	returned
11	832	69	33	60.24	10	returned
12	1004	195	32	49.97	10	returned
13	1010	198	3	48.64	10	returned
14	1049	223	20	15.66	10	returned
15	1171	298	50	88.19	10	returned
16	1422	471	36	47.01	10	returned
17	1349	419	12	66.93	5	returned
18	1355	424	10	84.26	5	returned

Inner Join/ Alias

SELECT * → DISTINCT → Top () * → From → Join → Where → Group By → Having → Order By

The screenshot displays the Microsoft SQL Server Enterprise Manager interface. On the left, the Object Explorer shows the database structure, including tables like `dbo.line_items` and `dbo.orders`. The main SQL Editor window contains a query that demonstrates an inner join between `line_items` and `orders`, filtered by a total value greater than \$700. The query is as follows:

```
-- EXERCISES: Answer using the techniques from above.
--
-- Let's find the name and email of people who purchased a line_item worth $700 or more.
-- We'll walk you through building up the query over several steps.
--
-- 1. Select everything from the line_items table.
-- In the results, notice it contains the price and quantity, but not who ordered it.
select * from line_items

-- 2. Join the line_items table to the orders table (on the order_id column)
-- In the results, notice you can now see price, quantity, and user_id.
select * from line_items li join orders o on o.order_id = li.order_id

-- 3. We want to get the user's name, so continuing with the previous query,
-- also join in the users table (on the user_id column).
-- In the results, notice you can now see the price, quantity, and name of the user.
select * from line_items li join orders o on o.order_id = li.order_id join users u on u.user_id = o.user_id

-- 4. Continuing with the previous query, filter the results to show
-- just the name and email of people who had a line_item of $700 or more.
-- Don't forget that involves math for quantity times price.
select name, email from line_items li join orders o on o.order_id = li.order_id join users u on u.user_id = o.user_id where li.price * li.quantity >= 700.00
```

The Results pane at the bottom shows the output of the query, displaying columns `name` and `email` for 17 rows of data. The status bar at the bottom indicates that the query was executed successfully, returning 22 rows.

Outer Joins/ Nulls

SELECT * → DISTINCT → Top () * → From → Join → Where → Group By → Having → Order By

FYIs

1. SQL Server case is insensitive

SQL Query Written Order



SQL	Purpose
SELECT	Specify the columns to show in result set
DISTINCT	Eliminate duplicate rows
TOP (SQL Server)	Limit the returned data to a specific number of rows
FROM	Get the base data from a table
JOIN	Obtain matching data from other table(s)
WHERE	Filter the base data
GROUP BY	Aggregate the base data (collect into groups)
HAVING	Filter the aggregated (grouped) data
ORDER BY	Sort the final data
LIMIT (Postgres)	Limit the returned data to a specific number of rows

2. Logical operators are used after a "WHERE" clause

Operator	Description
AND	Requires both specified conditions are met (true) for a record to be included in the result.
OR	Requires at least one of the specified conditions are met (true) for the record to be included in the result
NOT	Selects rows for the result which do not meet the specified criteria.

Instead of using "OR" clause, we can use "IN" clause

```
SELECT * FROM orders
```

```
WHERE ship_state = 'FL' OR ship_state = 'TX';
```

```
WHERE ship_state IN ('FL', 'TX');
```

```
SELECT * FROM products WHERE NOT title LIKE '%rubber%';
```

3. Alias- "AS" clause is optional when it comes to abbreviating table or column

Syntax for Aliases

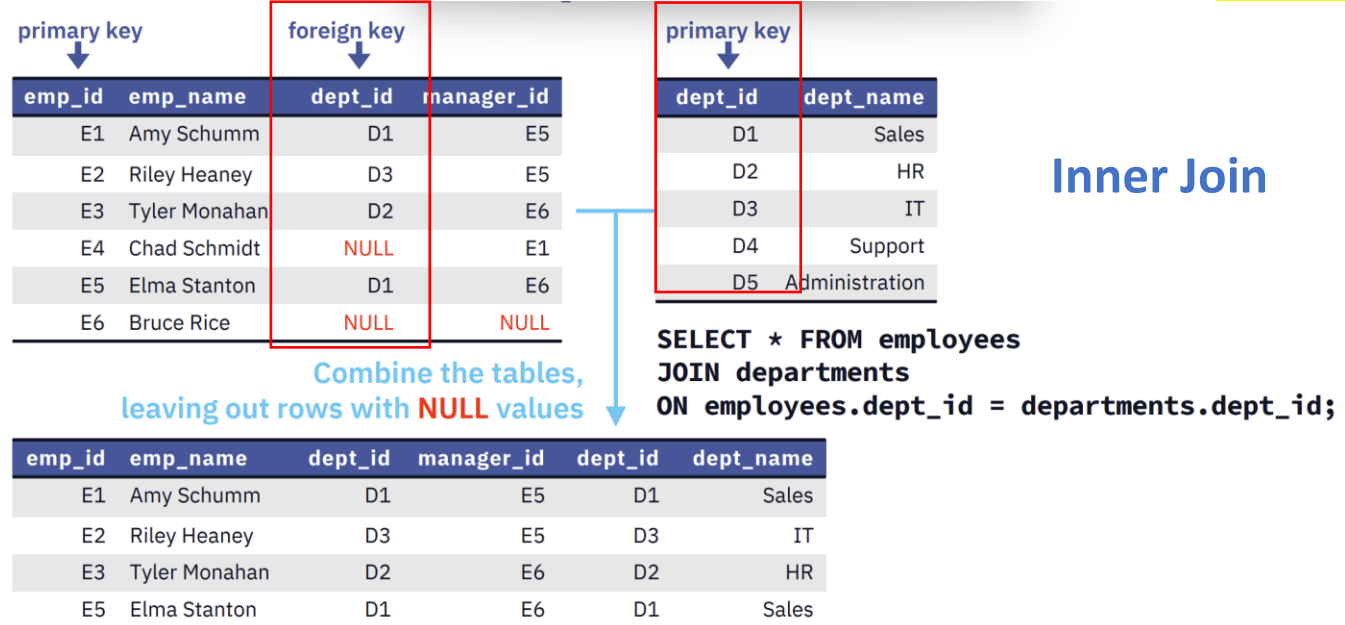
COLUMN ALIASES:

- emp_name AS Name
- emp_name Name
- emp_name AS "Employee Name"
- emp_name "Employee Name"

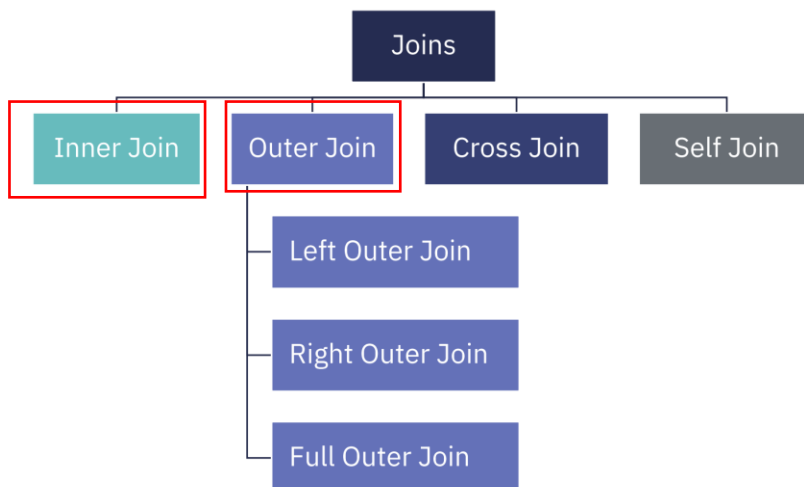
TABLE ALIASES:

- employees AS e
- employees e

4. A **foreign key** in one table, refers to a primary key in another table. However, 1 table can only have 1 **primary key** (aka unique identifier) at the same time. When multiple fields are used as a primary key, it is called a **composite key**.



5. A join combines data from multiple tables with or without using primary/foreign keys.



INNER JOIN: returns matching rows through the primary key (aka unique identifier)

OUTER JOIN: returns matching and non-matching rows (missing values appear as NULL).

The **Left, Right, Outer Joins** are all the same. They simply change the side where we keep unmatched rows. See below.

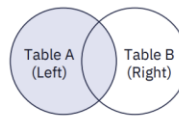
Outer Left Join

primary key				foreign key		primary key	
emp_id	emp_name	dept_id	manager_id	dept_id	dept_name	dept_id	dept_name
E1	Amy Schumm	D1	E5	D1	Sales	D1	Sales
E2	Riley Heaney	D3	E5	D2	HR	D2	HR
E3	Tyler Monahan	D2	E6	D3	IT	D3	IT
E4	Chad Schmidt	NULL	E1	D4	Support	D4	Support
E5	Elma Stanton	D1	E6	D5	Administration	D5	Administration
E6	Bruce Rice	NULL	NULL				

emp_id	emp_name	dept_id	manager_id	dept_id	dept_name
E1	Amy Schumm	D1	E5	D1	Sales
E2	Riley Heaney	D3	E5	D3	IT
E3	Tyler Monahan	D2	E6	D2	HR
E4	Chad Schmidt	NULL	E1	NULL	NULL
E5	Elma Stanton	D1	E6	D1	Sales
E6	Bruce Rice	NULL	NULL	NULL	NULL

**SELECT * FROM employees e
LEFT JOIN departments d
ON e.dept_id = d.dept_id;**

Combine the tables, keeping all rows from the left (first) table.

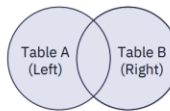


Full Outer Join

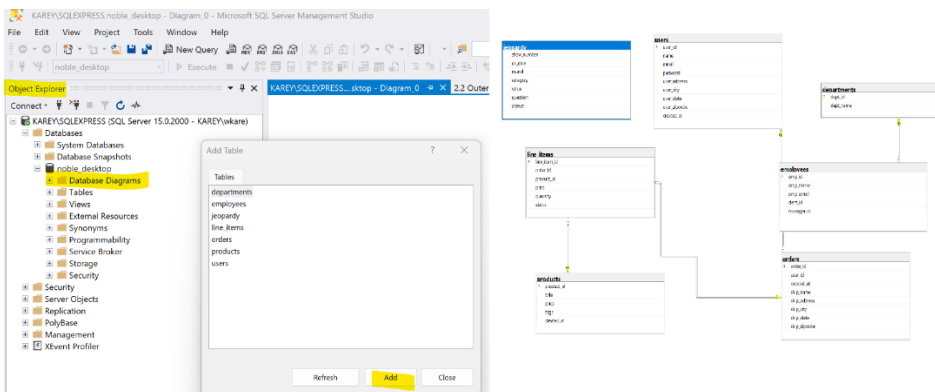
emp_id	emp_name	dept_id	manager_id	dept_id	dept_name
E1	Amy Schumm	D1	E5	D1	Sales
E2	Riley Heaney	D3	E5	D3	IT
E3	Tyler Monahan	D2	E6	D2	HR
E4	Chad Schmidt	NULL	E1	NULL	NULL
E5	Elma Stanton	D1	E6	D1	Sales
E6	Bruce Rice	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	D5	Administration
NULL	NULL	NULL	NULL	D4	Support

**SELECT * FROM employees e
FULL JOIN departments d
ON e.dept_id = d.dept_id;**

Combine the tables, keeping all rows from both tables.



6. Object Explorer → Right Click Database → Create Database Diagram → Yes → Add → Close



Schema

- A database schema is a set of tables.
- The schema defines how data is organized, the relations among tables.

