Database Design

SQL Essentials
To Query Databases

OVERVIEW OF THE FIVE DATABASE SESSIONS

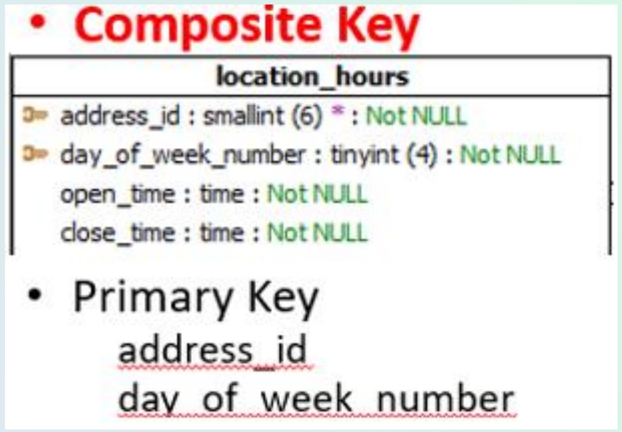
- Session 1: The Transactional Relational Database
  - Work product: Conceptual Model
- Session 2: Normalizing the Transactional Relational Database
  - Work product: Logical Model
- Session 3: Defining Data Structures Specific To A Database Platform (MariaDB)
  - Work product: Physical Model
- Session 4: Database Initialization Scripts To Create Databases & Objects
  - Work product: SQL Scripts To Create Database Objects
- **Session 5: SQL Essentials To Query Databases**
  - **Work product: SQL Commands To Query Databases**

First, Let's Review

- What is the unique identifier of a table?
  - **The Primary Key**
- Which model shows the entities related by primary and foreign keys?
  - **The Logical Model**
- What type of constraint enforces specific values for a column?
  - **A Check Constraint**
- Which model is dependent on which database is being used?
  - **The Physical Model**
- What types of keys enforce referential integrity?
  - **The Primary Keys and Foreign Keys**
- Which model must be understood by everyone?
  - **The Conceptual Model**

First, Let's Review

- What type of Primary Key includes multiple columns?
  - A Composite Key
- Which type of Primary Key includes only 1 column?
  - A Surrogate Key
- **Examples: Would both accomplish the same function?** Yes

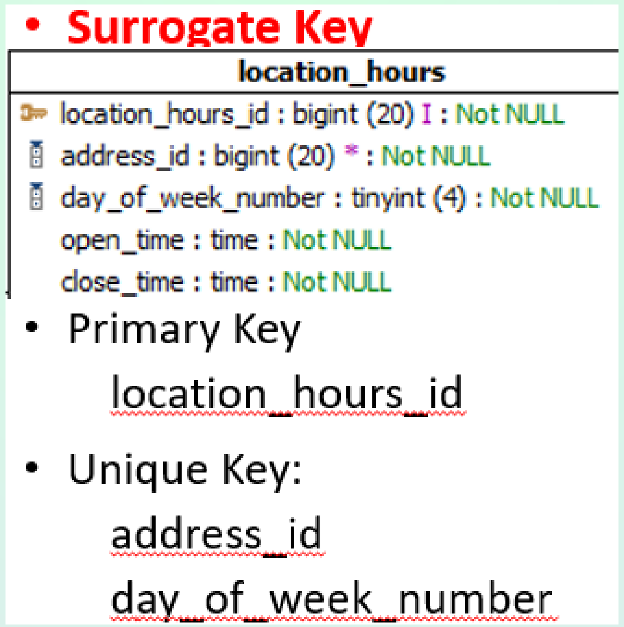


- Composite Key

location_hours

address_id : smallint (6) * : Not NULL
day_of_week_number : tinyint (4) : Not NULL
open_time : time : Not NULL
close_time : time : Not NULL

- Primary Key
  address_id
  day_of_week_number



- Surrogate Key

location_hours

location_hours_id : bigint (20) I : Not NULL
address_id : bigint (20) * : Not NULL
day_of_week_number : tinyint (4) : Not NULL
open_time : time : Not NULL
close_time : time : Not NULL

- Primary Key
  location_hours_id
- Unique Key:
  address_id
  day_of_week_number

Session 5 Objectives

- What is a SQL query
- The SELECT statement
- Simple SQL Queries
- Column & Table Aliases
- Common SQL Functions
- Filtering Query Results
- Helpful SQL Commands
- Create SQL Toolbox
- Using indexes to improve performance

SQL Queries

- What is a SQL query?
    - It is a **SELECT** statement that queries a database and returns the requested data.
    - It can be used to:
        - Look at data from a single table
        - Combine data from multiple tables
        - Filter the data returned
        - Format the data returned
        - Sort the data returned
        - Summarize the data returned

The SELECT statement

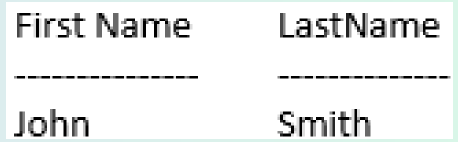
- What is Required:
  - **SELECT** {*Add column names of the data that you want to see*}
  - **FROM** {*Add the tables that store the data*}
- Optional:
  - **WHERE** {*Specify the criteria/conditions for the data to be included*}
  - **ORDER BY** {*Specify the order to display the data*}
- Required – When summarizing/calculating/grouping data:
  - **GROUP BY** {*Specify how to summarize/group the data*}
- Optional – When summarizing/calculating/grouping data:
  - **HAVING** {*Specify the criteria/conditions for the data to be included*}

Simple SQL Queries

- SELECT * FROM customer;
  - This is a quick way to take a **1**st look at data in a table.
  - This may be quicker and less typing, however, it has risks:
    - What would happen if a column was ***added to*** the table?
    - What would happen if a column was ***removed from*** the table?
    - What would happen if the ***column order was changed***?
  - This should not be used in your code!
- SELECT customer_id, first_name, last_name FROM customer;
  - This is a "Best Practice" coding habit to get into.
    - This reduces the risks of your code breaking.
      - It **is not impacted** if a column is ***added to*** the table.
      - It **is not impacted** if the ***column order is changed.***
      - It **is impacted** ONLY if a column that ***is selected*** is ***removed from*** the table.

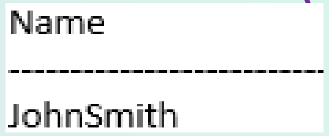
Simple SQL Queries – Aliases are helpful

- What is an Alias?
  - It is an alternative column name or table name
- SELECT first_name AS 'First Name', last_name AS LastName FROM customer;



```
First Name        LastName
---------------   -------------
John              Smith
```

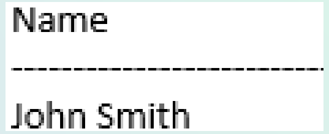
- What if the requirement want to see the names in a single column?
  - SELECT CONCAT(first_name, last_name) AS Name FROM customer;



```
Name
-------------------------
JohnSmith
```

(Is this ok?)

  - SELECT CONCAT(first_name, ' ', last_name) AS Name FROM customer;



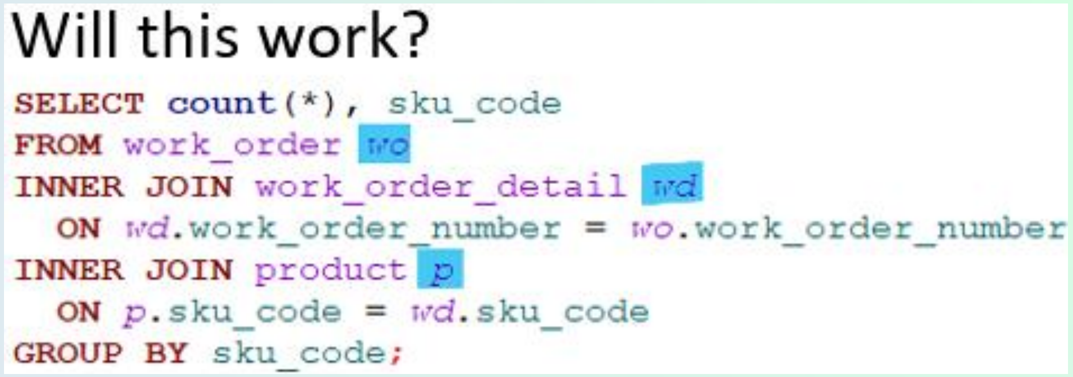
```
Name
-------------------------
John Smith
```

(Is this better?)

- These are examples of using a column alias.

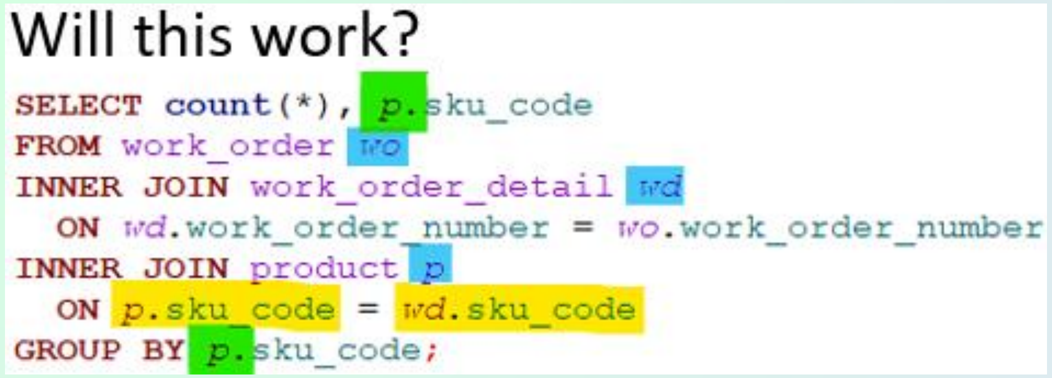
Complex SQL Queries – Table Aliases

- Use table aliases when joining more than 1 table.
  - Without a table alias, join conditions must be fully qualified with table names.
- Here are examples of using a table alias.



Will this work?

```
SELECT count(*), sku_code
FROM work_order wo
INNER JOIN work_order_detail wd
  ON wd.work_order_number = wo.work_order_number
INNER JOIN product p
  ON p.sku_code = wd.sku_code
GROUP BY sku_code;
```

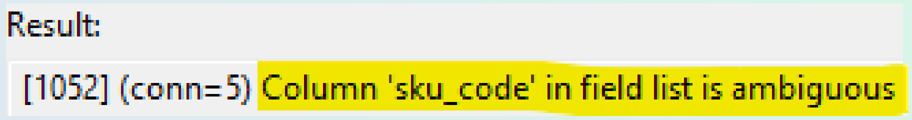


Will this work?

```
SELECT count(*), p.sku_code
FROM work_order wo
INNER JOIN work_order_detail wd
  ON wd.work_order_number = wo.work_order_number
INNER JOIN product p
  ON p.sku_code = wd.sku_code
GROUP BY p.sku_code;
```

This will cause an **Error**.

This will run **Successfully**.



```
Result:
[1052] (conn=5) Column 'sku_code' in field list is ambiguous
```

Using Common SQL Functions

- SQL Functions allow you to summarize, calculate, or group your data
  - **Count(*)** – counts the total number of rows returned by the query

    ```
    SELECT COUNT(*) FROM work_order;
    ```

    {*this will count the total number of work orders*}
  - **SUM** – calculates the total of value of the rows returned

    ```
    SELECT SUM(unit_cost) FROM work_order_detail WHERE work_order_number = 118571;
    ```

    {*this will calculate the total cost for the Work Order*}
  - **MAX** – returns the largest or maximum number

    ```
    SELECT sku_code, MAX(quantity) AS NbrSold
    FROM work_order_detail
    GROUP BY sku_code
    ORDER BY NbrSold desc;
    ```

    {*this will show the max quantity of each item that was sold. It is also ordered by the NbrSold descending so the item that was sold the most will be returned first*}

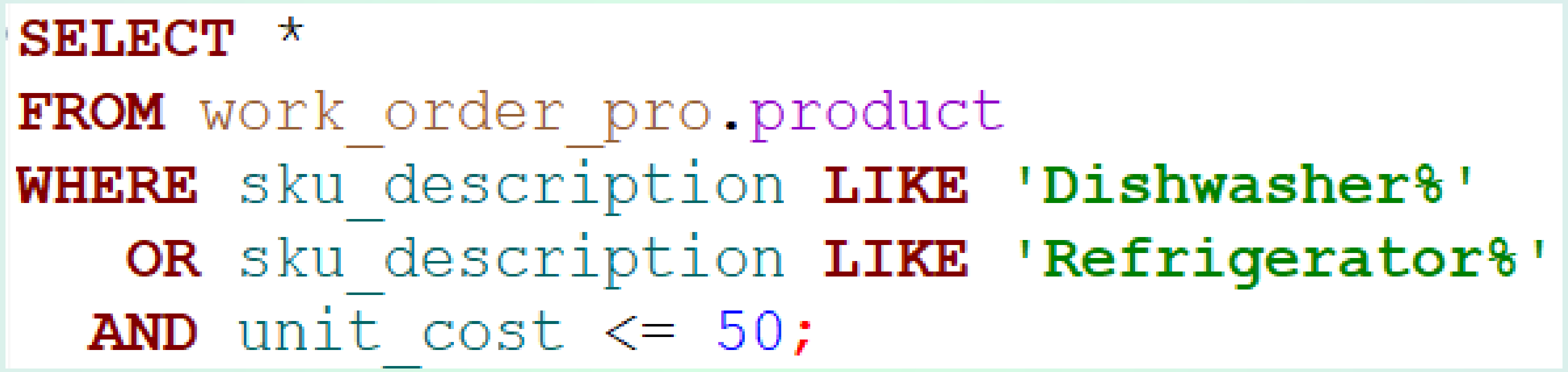
Filtering Query Results

- WHERE clause (with comparison operators)
  - Date > ‘2024-09-11’
    - Common Operators include >, <, =, between {date} and {date}
  - Text = ‘Work Order Pro’
    - Common Operators include =, !=, <>, >, <, like, not like, in, not in
- AND / OR (be careful when using a combination of both of these)
  - Parenthesis matter
- LIKE (can use % or _ as wildcard)
- LIMIT
- ORDER BY clause

Filtering Query Results – Using AND / OR

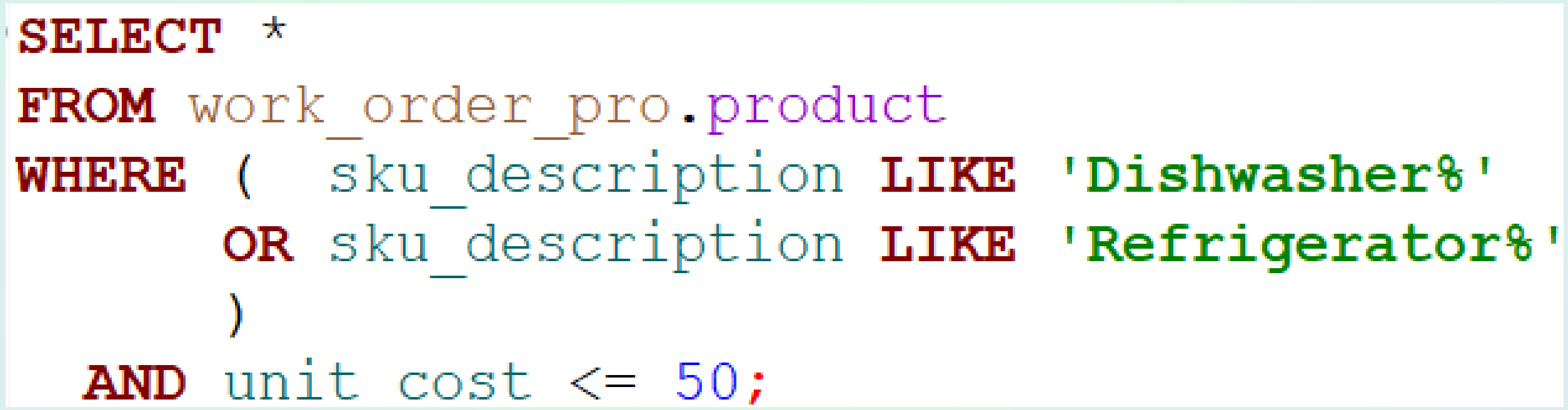
List all Dishwasher or Refrigerator products that cost less than or equal $50.

OPTION A:



```sql
SELECT *
FROM work_order_pro.product
WHERE sku_description LIKE 'Dishwasher%'
   OR sku_description LIKE 'Refrigerator%'
  AND unit_cost <= 50;
```

OPTION B:



```sql
SELECT *
FROM work_order_pro.product
WHERE (   sku_description LIKE 'Dishwasher%'
      OR sku_description LIKE 'Refrigerator%'
      )
  AND unit_cost <= 50;
```

Which Option Is Correct?   LET'S CHECK

Helpful SQL commands

- **DESCRIBE (**DESCRIBE customer; **)** {*to display the table definition*}
- **SELECT DISTINCT** {*to display results without duplicate rows*}
- **SELECT COUNT(*)** {*to count the occurrences of something*}
- **SELECT SUM(column1)** {*to calculate the total value of a column*}
- **CONCAT(value1, value2, etc)** {*to put values together*}
- **GROUP BY** {*used to summarize data*}

SQL Toolbox

- It is an organized, shared, common area utilized within work groups.
- It should contain:
  - Any SQL queries written to answer common business questions.
    - *How many work orders did we have last month?*
  - Any SQL scripts written to extract data for business requests.
    - *Please pull a list of customers in Ohio.*
  - Any SQL scripts used for researching data issues.
    - *Identify rows with invalid numbers that should be used in a calculation*
    - *{For example, someone scanned a barcode number into the quantity field}*
- It is very convenient when trying to resolve Production Issues after normal business hours.

Using Indexes To Improve Performance

- Think of a database index like the index in the back of a textbook.
- Indexes are one of the most important and most impacting ways to maximize fast performance in a relational database.
- An index can be created on one or more columns in a table.
- Indexes can vastly reduce the amount of time it takes for queries or normal routine processes to complete.
- Without indexes, the database has to perform a full table scan in order to locate the requested data.
  - *This is very slow on tables with thousands of rows of data.*
- ***Caution:*** *Too many indexes can slow down inserts, updates, and deletes.*

What Columns To Create An Index On:

- We get a lot of requests to pull customer lists **by state**.

- So, the **state** would be a good choice for an index to improve the performance of this request.

- SQL Commands To Create or Drop indexes:

```
CREATE INDEX customer_state_idx ON customer (state);
DROP INDEX customer_state_idx;
```

Thank you for being great students.

Hopefully you learned a lot.

ENJOY THE REST OF YOUR CLASS!