Session 2 – Manual

Data Manipulation

Welcome to the SQL Manual for Filtering, Sorting, and updating Data. In this session, we'll delve deeper into the techniques used to manipulate data within SQL databases effectively.

Objective:

• **Understand the WHERE Clause:**

- Learn to filter rows based on specified conditions.
- Explore various condition types, including comparisons, logical operators, and pattern matching.

• **Master the ORDER BY Clause:**

- Discover how to sort query results by specified columns in ascending or descending order. Understand the syntax and key points for using multiple columns in sorting.
- Implement the ORDER BY clause with practical examples.

• **Utilize the LIMIT Clause:**

- Learn to restrict the number of rows returned by a query.
- Apply the LIMIT clause in queries to manage large datasets effectively.

• **Modifying and Updating New Tables**:

- Modify the structure of existing tables using the `ALTER TABLE` statement
- Update data within these tables using the `UPDATE` statement. This includes adding new columns, changing data types, and updating existing records.

• **Understand and Implement Primary Keys**:

- Define and create primary keys to uniquely identify records in a table.
- Add and remove primary keys from existing tables.

• **Utilize Auto-Increment Fields**:

- Configure auto-increment fields to automatically generate unique identifiers for new records.
- Reset auto-increment values as needed.

1. WHERE Clause:

The WHERE clause filters rows from a table based on a specified condition, allowing you to retrieve only the data that meets certain criteria.

Syntax:

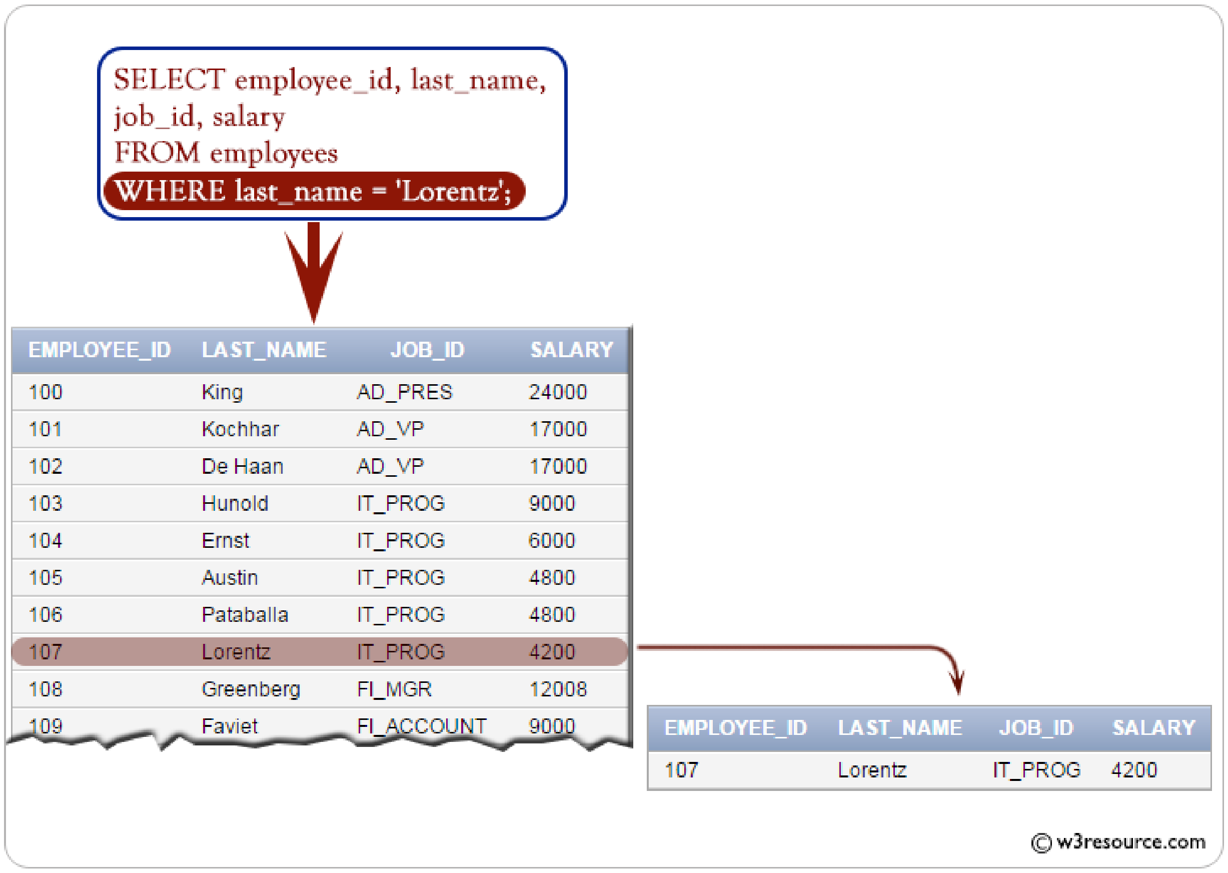
SELECT column1, column2, ...
FROM table_name
WHERE condition;

Example:

SELECT * FROM employees WHERE department = 'Sales';

Key Points:

- Conditions in the WHERE clause can include comparisons (=, !=, >, <, >=, <=), logical operators (AND, OR, NOT), and pattern matching using LIKE.
- Use single quotes for string values and no quotes for numeric values in conditions.



```
SELECT employee_id, last_name,
job_id, salary
FROM employees
WHERE last_name = 'Lorentz';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000
105	Austin	IT_PROG	4800
106	Pataballa	IT_PROG	4800
107	Lorentz	IT_PROG	4200
108	Greenberg	FI_MGR	12008
109	Faviet	FI_ACCOUNT	9000

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
107	Lorentz	IT_PROG	4200

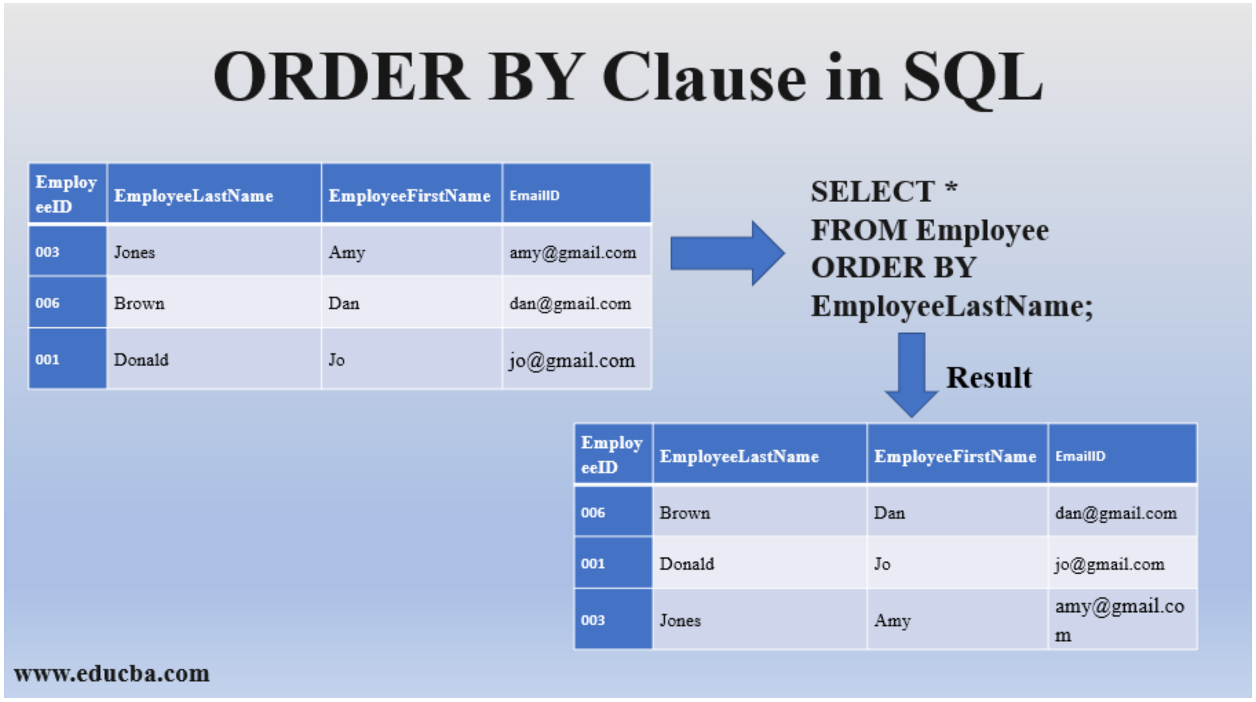
2. ORDER BY Clause:

The ORDER BY clause sorts the result set of a query based on specified columns, either in ascending or descending order.

Syntax:

SELECT column1, column2, ...

FROM table_name
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;



EmployeeID	EmployeeLastName	EmployeeFirstName	EmailID
003	Jones	Amy	amy@gmail.com
006	Brown	Dan	dan@gmail.com
001	Donald	Jo	jo@gmail.com

```
SELECT *
FROM Employee
ORDER BY
EmployeeLastName;
```

EmployeeID	EmployeeLastName	EmployeeFirstName	EmailID
006	Brown	Dan	dan@gmail.com
001	Donald	Jo	jo@gmail.com
003	Jones	Amy	amy@gmail.com

Example:

SELECT * FROM products ORDER BY price DESC;

Key Points:

- You can specify multiple columns for sorting, with the order of precedence determined by the sequence of columns in the ORDER BY clause.
- Use ASC for ascending order (default) and DESC for descending order.

3. LIMIT Clause:

The LIMIT clause is used to restrict the number of rows returned by a query, which is particularly useful when dealing with large datasets.

Syntax:

SELECT column1, column2, ...

FROM table_name
LIMIT number_of_rows;

Example:

SELECT * FROM customers LIMIT 10;

Key Points:

- The LIMIT clause is not supported by all SQL databases, so check the documentation of your specific database management system (DBMS) for compatibility.
- LIMIT is often used in combination with ORDER BY to retrieve the top or bottom N records based on a specified criterion.

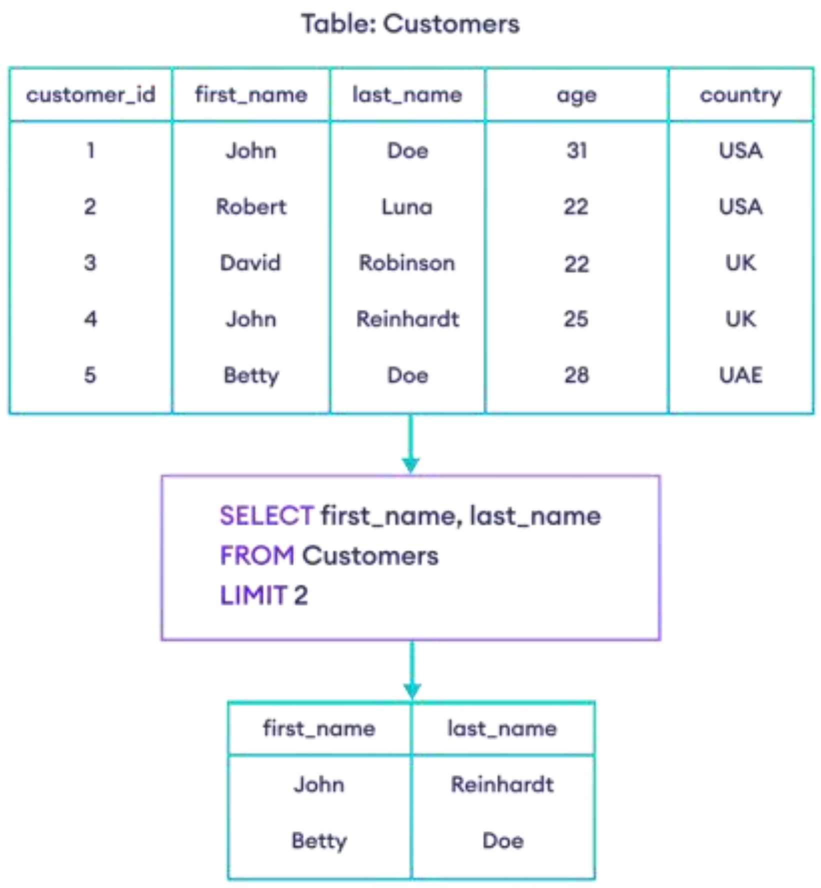


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
LIMIT 2
```

first_name	last_name
John	Reinhardt
Betty	Doe

4. Modifying and Updating New Tables

You can modify the structure of an existing table using the `ALTER TABLE` statement and update the data using the `UPDATE` statement.

Altering Table Structure

Syntax

```
ALTER TABLE table_name
ADD column_name datatype;
```

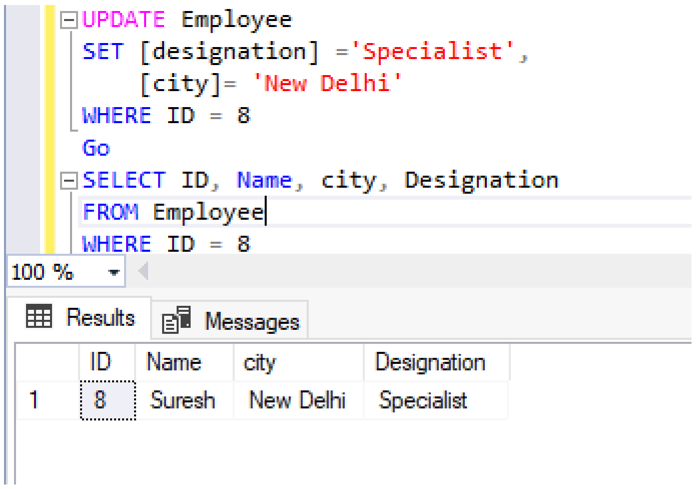
Example

Add a new column `record_date` to the `users` table.

```
ALTER TABLE users
ADD record_date Date;
```

Updating Data

General Example of updating data in table.



```
UPDATE Employee
SET [designation] ='Specialist',
    [city]= 'New Delhi'
WHERE ID = 8
Go
SELECT ID, Name, city, Designation
FROM Employee
WHERE ID = 8
```

	ID	Name	city	Designation
1	8	Suresh	New Delhi	Specialist

Syntax

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

Example

Update the salary of the employee with `employee_id` 1.

```
UPDATE employees
SET salary = 80000.00
WHERE employee_id = 1;
```

5. Working with Primary Keys

Introduction

A primary key is a field in a table which uniquely identifies each row/record in that table. Primary keys must contain unique values and cannot contain NULL values.

Creating a Primary Key

To create a primary key when creating a table, use the following syntax:

```
CREATE TABLE table_name (
    column1 datatype PRIMARY KEY,
    column2 datatype,
    ...
);
```

Example

```
CREATE TABLE Students (
    StudentID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Adding a Primary Key to an Existing Table

If a table already exists and you want to add a primary key:

```
ALTER TABLE table_name
ADD PRIMARY KEY (column_name);
```

Example

```
ALTER TABLE Students
ADD PRIMARY KEY (StudentID);
```

Removing a Primary Key

To remove a primary key constraint:

```
ALTER TABLE table_name
DROP PRIMARY KEY;
```

Example

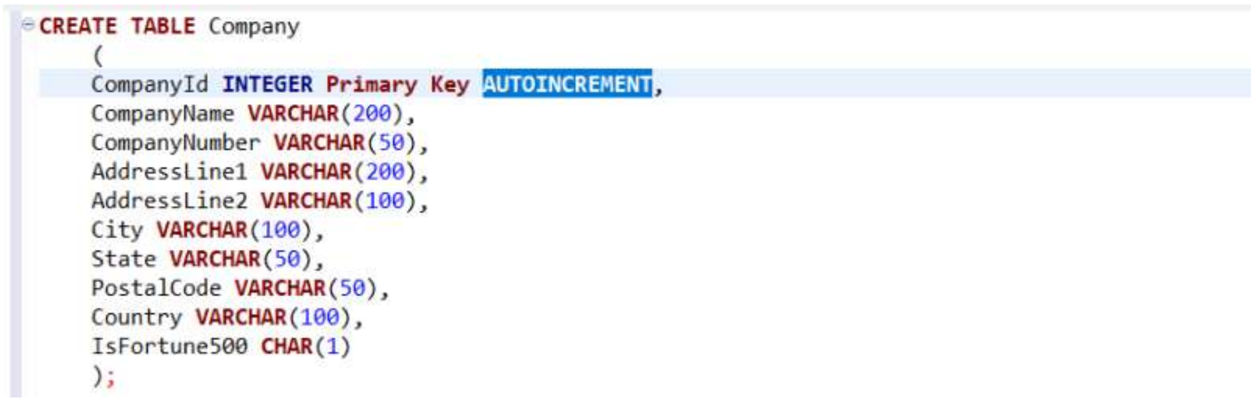
```
ALTER TABLE Students
DROP PRIMARY KEY;
```

6. Auto-Increments

Introduction

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

General Example of auto increment.



```
CREATE TABLE Company
    (
    CompanyId INTEGER Primary Key AUTOINCREMENT,
    CompanyName VARCHAR(200),
    CompanyNumber VARCHAR(50),
    AddressLine1 VARCHAR(200),
    AddressLine2 VARCHAR(100),
    City VARCHAR(100),
    State VARCHAR(50),
    PostalCode VARCHAR(50),
    Country VARCHAR(100),
    IsFortune500 CHAR(1)
    );
```

Creating an Auto-Increment Field

To create an auto-increment field, you can use the `AUTO_INCREMENT` attribute in MySQL or the `SERIAL` data type in PostgreSQL.

MySQL Example

```
CREATE TABLE Students (
    StudentID int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (StudentID)
);
```

Resetting Auto-Increment Value

To reset the auto-increment value in MySQL:

```
ALTER TABLE table_name AUTO_INCREMENT = value;
```

Example

```
ALTER TABLE Students AUTO_INCREMENT = 1000;
```

Best Practices

- Always use the `WHERE` clause to specify which records to update.
- Test your `UPDATE` statements with a `SELECT` query first.

Remember to always end your SQL statements with a semicolon (;) and to use appropriate whitespace and indentation for readability. Utilize these techniques to efficiently filter, sort, and update data in your SQL databases. Happy querying!