Introduction

This guide provides a formal, point-by-point overview of essential MySQL topics. It is designed as a quick reference document, presenting key concepts, syntax, and basic examples in a clear and concise manner for easy understanding.

1. Table Creation & Constraints (DDL basics)

- **Purpose:** Data Definition Language (DDL) is used to create and modify the structure of database objects, such as tables.[1] These commands define the blueprints and rules for your data.[3]
- **CREATE TABLE Syntax:**
  SQL
  ```
  CREATE TABLE table_name (
    column1 datatype [constraints],
    column2 datatype [constraints],
   ...
  );
  ```

  The IF NOT EXISTS clause prevents an error if the table already exists.[5]
- **Constraints:** Constraints are rules that enforce data integrity and quality.[7]
  - **PRIMARY KEY:** A column that uniquely identifies each row in a table. It must contain unique and non-null values.[7] A table can have only one primary key.
  - **FOREIGN KEY:** A column in one table that links to the PRIMARY KEY of another table to establish a relationship.[9]
  - **NOT NULL:** Ensures a column cannot contain an empty (NULL) value.[7]
  - **UNIQUE:** Guarantees that all values in a column or set of columns are distinct.[7] Unlike a
    PRIMARY KEY, it can have multiple NULL values.
  - **CHECK:** Validates that data being inserted or updated meets a specific condition.[7]
- **Example:**
  SQL
  ```
  CREATE TABLE supplies (
    supply_id INT PRIMARY KEY,
    name VARCHAR(250) NOT NULL,
    inventory INT,
  ```

```
  CHECK (inventory >= 0)
);
```

This command creates a table named supplies with a PRIMARY KEY on supply_id, a NOT NULL constraint on the name column, and a CHECK constraint to ensure inventory is never a negative number.[7]

2. Data Manipulation (DML basics)

- **Purpose:** Data Manipulation Language (DML) is used to add, change, and delete data *within* the tables that were built with DDL. These operations are transactional and can be rolled back.
- **INSERT:** Adds new rows of data into a table.
  - **Syntax:**
    SQL
    ```
    INSERT INTO table_name (column1, column2)
    VALUES (value1, value2);
    ```
  - Multiple rows can be inserted in a single command for efficiency.[11]
- **UPDATE:** Modifies existing data in a table.[13]
  - **Syntax:**
    SQL
    ```
    UPDATE table_name
    SET column1 = new_value
    WHERE condition;
    ```
  - **Note:** A WHERE clause is critical to specify which rows to change, preventing a full table update.[11]
- **DELETE:** Removes existing rows from a table.[13]
  - **Syntax:**
    SQL
    ```
    DELETE FROM table_name
    WHERE condition;
    ```
  - **Note:** Always use a WHERE clause to avoid deleting all records.[11]
- **Example:**
  SQL
  ```
  -- Insert a new row into a People table
  INSERT INTO People (id, name, occupation)
  ```

```sql
VALUES (105, 'Chris Green', 'Analyst');

-- Update an existing record
UPDATE People
SET occupation = 'Senior Analyst'
WHERE id = 105;

-- Delete a record
DELETE FROM People
WHERE id = 105;
```

3. Querying Data

- **Purpose:** The SELECT statement is the primary command for retrieving data from one or more tables.[15]
- **Syntax:**
  SQL
  ```sql
  SELECT column1, column2
  FROM table_name
  WHERE condition
  GROUP BY column
  HAVING condition
  ORDER BY column;
  ```
- **SELECT:** Specifies which columns to retrieve. Use SELECT * to retrieve all columns.[15]
- **FROM:** Specifies the table from which to retrieve data.[15]
- **WHERE:** An optional clause to filter individual rows based on a specified condition.[15]
- **GROUP BY:** Groups rows with the same values into summary rows, typically used with aggregate functions like COUNT() or SUM().[15]
- **HAVING:** Filters the groups created by GROUP BY. Unlike WHERE, it can apply conditions to aggregate functions.[20]
- **ORDER BY:** Sorts the final result set in ascending (ASC) or descending (DESC) order.[15]
- **Example:**
  SQL
  ```sql
  -- Query for employees with a salary greater than 50,000,
  -- grouped by department and sorted by salary in descending order
  SELECT department, SUM(salary) AS total_salary
  FROM employees
  WHERE salary > 50000
  ```

```
GROUP BY department
HAVING total_salary > 100000
ORDER BY total_salary DESC;
```

This query first filters out employees with a salary of 50,000 or less, then calculates the total salary for each remaining department. The HAVING clause then filters to show only departments where the total salary is over 100,000, and the final result is sorted by the total salary from highest to lowest.[15]

4. Joins (Connecting tables)

- **Purpose:** Used to combine data from two or more tables into a single result set based on a related column.[22]
- **INNER JOIN:** Returns only the rows that have matching values in **both** tables.[24]
    - **Syntax:** SELECT... FROM table1 INNER JOIN table2 ON table1.column = table2.column;.[25]
- **LEFT JOIN:** Returns all rows from the **left** table and the matching rows from the right table. If there is no match, it returns NULL values for the right table's columns.[24]
- **RIGHT JOIN:** Returns all rows from the **right** table and the matching rows from the left table. If there is no match, it returns NULL values for the left table's columns.[24]
- **CROSS JOIN:** Combines every single row from the first table with every single row from the second table.[24] It does not require a join condition.[27]
- **Example:**

  SQL
  ```
  -- INNER JOIN example to find employees and their department names
  SELECT
    employees.name,
    departments.department_name
  FROM
    employees
  INNER JOIN
    departments ON employees.department_id = departments.department_id;
  ```

  This query retrieves the name of each employee and the name of their corresponding department, but only for employees who are actually assigned to a department.[22]

5. Subqueries

- **Purpose:** A subquery is a query nested inside another query, used to perform operations in multiple steps or to act as input for the outer query. They are always enclosed in parentheses.[28]
- **Types:**
  - **Non-correlated:** A subquery that is independent of the outer query. It executes first and returns its result to the outer query.
  - **Correlated:** A subquery that is dependent on the outer query. It references a column from the outer query and is executed once for each row processed by the outer query.[6] This can impact performance.
- **Example:**
  ```sql
  -- Find all products that have a price higher than the average product price
  SELECT
    name,
    price
  FROM
    products
  WHERE
    price > (SELECT AVG(price) FROM products);
  ```

  The inner query (SELECT AVG(price) FROM products) runs first to find the average price, and its single-value result is then used by the outer query to filter the products.[36]

6. Indexes

- **Purpose:** An index is a data structure that improves the speed of data retrieval operations, similar to a book index.[27] It allows MySQL to quickly find specific rows without scanning the entire table.[19]
- **Syntax:**
  ```sql
  CREATE INDEX index_name
  ON table_name (column_name);
  ```
- **Key Details:**
  - While indexes speed up SELECT queries, they can slow down INSERT, UPDATE, and DELETE operations because the index must also be updated when data changes.
  - An index is automatically created when a PRIMARY KEY is defined on a table.[27]
- **Example:**

```sql
-- Create an index on the email column in a users table
CREATE INDEX email_idx
ON users (email);
```

This index allows the database to quickly find a user's record when a query filters on the email column, avoiding a full table scan.[37]

7. Triggers

- **Purpose:** A trigger is a set of SQL statements that automatically executes in response to a specific event on a table, such as INSERT, UPDATE, or DELETE. They are used to automate tasks and enforce data consistency.
- **Syntax:**
  ```sql
  DELIMITER //
  CREATE TRIGGER trigger_name
  BEFORE | AFTER INSERT | UPDATE | DELETE
  ON table_name
  FOR EACH ROW
  BEGIN
     -- SQL statements
  END; //
  DELIMITER ;
  ```
  - The DELIMITER command is used to change the statement end character, allowing for multiple commands within the trigger's body.[32]
- **OLD and NEW Keywords:** These keywords refer to the data in the row *before* (OLD) and *after* (NEW) the operation.
- **Example:**
  ```sql
  -- Create a trigger to automatically send a welcome email after a new customer is inserted
  DELIMITER //
  CREATE TRIGGER after_customer_insert
  AFTER INSERT ON customers
  FOR EACH ROW
  BEGIN
     INSERT INTO email_queue (customer_id, email_type, status)
     VALUES (NEW.id, 'welcome', 'pending');
  END; //
  ```

```sql
DELIMITER ;
```

This trigger automatically inserts a record into an email_queue table every time a new row is added to the customers table, automating a task without requiring an external application to do so.[17]

8. Stored Procedures & Functions

- **Purpose:** Both are pre-written blocks of SQL code that are saved on the database server to be reused.[12] This improves performance, security, and maintainability.[12]
- **Stored Procedures:**
  - Used to perform an action or execute business logic.[12]
  - Can return multiple values or result sets.[12]
  - Can modify the database state.[12]
  - **Invocation:** CALL procedure_name();.[12]
- **Stored Functions:**
  - Used to compute a single value.[12]
  - Must return a single, scalar value.[12]
  - Generally cannot modify the database state.[12]
  - **Invocation:** Can be used directly within a SELECT statement.[12]
- **Example:**
  SQL
  ```sql
  -- Stored Procedure Example: Get employees for a specific department
  DELIMITER //
  CREATE PROCEDURE GetEmployeesByDepartment (IN dept_name VARCHAR(50))
  BEGIN
    SELECT * FROM employees WHERE department = dept_name;
  END //
  DELIMITER ;

  -- Invoking the procedure:
  CALL GetEmployeesByDepartment('Marketing');
  ```

  This procedure retrieves all employees from a specified department.[12]
  SQL
  ```sql
  -- Stored Function Example: Calculate an employee's age
  DELIMITER //
  CREATE FUNCTION CalculateAge (birth_date DATE)
  RETURNS INT
  ```

```
DETERMINISTIC
BEGIN
  DECLARE today DATE;
  SELECT CURRENT_DATE() INTO today;
  RETURN YEAR(today) - YEAR(birth_date);
END //
DELIMITER ;

-- Using the function in a query:
SELECT name, CalculateAge(birth_date) AS age
FROM employees;
```

This function calculates an employee's age and returns it as a single value, which is then used as a column in the SELECT query.[25]

Conclusion

Mastering these topics provides a solid foundation for any data professional working with MySQL. From defining database structures with DDL to manipulating data with DML, this guide covers the core commands needed for effective data management and analysis. The ability to use joins, subqueries, and advanced features like indexes, triggers, and stored procedures will enable a data professional to write more efficient, secure, and powerful queries. This document serves as a practical starting point for continued learning and exploration.