MySQL Complete Notes

1. MySQL Engines

Storage engines determine how data is stored, indexed, and retrieved.

Common Engines:

- **InnoDB**: Default engine, supports transactions, foreign keys, row-level locking
- **MyISAM**: Faster for read-heavy operations, table-level locking, no transactions
- **Memory**: Stores data in RAM, fast but temporary
- **Archive**: For compressed, archival data

```sql
-- Check available engines
SHOW ENGINES;

-- Create table with specific engine
CREATE TABLE users (id INT PRIMARY KEY) ENGINE=InnoDB;
```

2. SELECT Statement

Used to retrieve data from tables.

```sql
-- Basic SELECT
SELECT column1, column2 FROM table_name;
SELECT * FROM table_name;

-- With conditions
SELECT * FROM employees WHERE salary > 50000;

-- With sorting
SELECT * FROM employees ORDER BY salary DESC;

-- With limiting results
SELECT * FROM employees LIMIT 10;

-- With multiple conditions
SELECT * FROM employees WHERE department = 'IT' AND salary > 60000;
```

3. INSERT Statement

Used to add new records to a table.

```sql
-- Insert single record
INSERT INTO employees (name, email, salary)
VALUES ('John Doe', 'john@email.com', 55000);

-- Insert multiple records
INSERT INTO employees (name, email, salary) VALUES
('Alice Smith', 'alice@email.com', 60000),
('Bob Johnson', 'bob@email.com', 52000);

-- Insert from another table
INSERT INTO employees_backup SELECT * FROM employees;
```

4. UPDATE Statement

Used to modify existing records.

```sql
-- Update specific records
UPDATE employees SET salary = 65000 WHERE id = 1;

-- Update multiple columns
UPDATE employees SET salary = 70000, department = 'Senior IT'
WHERE name = 'John Doe';

-- Update with conditions
UPDATE employees SET salary = salary * 1.1 WHERE department = 'IT';
```

5. DELETE Statement

Used to remove records from a table.

```sql
-- Delete specific records
DELETE FROM employees WHERE id = 1;

-- Delete with conditions
DELETE FROM employees WHERE salary < 30000;

-- Delete all records (keeps table structure)
DELETE FROM employees;
```

6. Primary Key and Foreign Key

Primary Key

- Uniquely identifies each record
- Cannot be NULL
- Only one per table

```sql
CREATE TABLE employees (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL
);

-- Or add after creation
ALTER TABLE employees ADD PRIMARY KEY (id);
```

Foreign Key

- Links two tables together
- Refers to primary key in another table
- Maintains referential integrity

```sql
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50)
);

CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(100),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```

7. Difference between InnoDB and MyISAM

Feature	InnoDB	MyISAM
Transactions	Yes (ACID compliant)	No
Foreign Keys	Yes	No
Locking	Row-level	Table-level
Crash Recovery	Yes	Limited
Storage Space	More	Less
Performance	Better for mixed read/write	Better for read-heavy
Full-text Search	Yes (5.6+)	Yes

8. Indexes

Indexes improve query performance by creating shortcuts to data.

Types:

- **Primary Index**: Automatically created for primary key
- **Unique Index**: Ensures uniqueness
- **Composite Index**: Multiple columns
- **Full-text Index**: For text searching

```sql
-- Create index
CREATE INDEX idx_salary ON employees(salary);

-- Composite index
CREATE INDEX idx_name_dept ON employees(name, department);

-- Unique index
CREATE UNIQUE INDEX idx_email ON employees(email);

-- Drop index
DROP INDEX idx_salary ON employees;

-- Show indexes
SHOW INDEXES FROM employees;
```

9. Difference between CHAR and VARCHAR

Feature	CHAR	VARCHAR
Storage	Fixed length	Variable length
Padding	Space-padded	No padding
Performance	Faster	Slightly slower
Storage Space	Uses full declared size	Uses actual data size + 1-2 bytes
Max Length	255 characters	65,535 characters

```sql
-- CHAR example
name CHAR(10)     -- Always uses 10 bytes

-- VARCHAR example
name VARCHAR(10)  -- Uses 1-10 bytes + length info
```

10. Aggregate Functions

Functions that perform calculations on multiple rows.

```sql
-- Common aggregate functions
SELECT COUNT(*) FROM employees;          -- Count rows
SELECT SUM(salary) FROM employees;       -- Sum values
SELECT AVG(salary) FROM employees;       -- Average
SELECT MAX(salary) FROM employees;       -- Maximum
SELECT MIN(salary) FROM employees;       -- Minimum

-- With grouping
SELECT department, COUNT(*), AVG(salary)
FROM employees GROUP BY department;
```

11. GROUP BY & HAVING

GROUP BY

Groups rows with same values in specified columns.

HAVING

Filters groups (use WHERE for individual rows).

```sql
```

```sql
-- GROUP BY example
SELECT department, COUNT(*) as emp_count, AVG(salary) as avg_salary
FROM employees
GROUP BY department;

-- HAVING example
SELECT department, COUNT(*) as emp_count
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;

-- Combined with WHERE
SELECT department, AVG(salary)
FROM employees
WHERE salary > 40000
GROUP BY department
HAVING AVG(salary) > 60000;
```

12. Difference between DELETE, DROP, and TRUNCATE

Operation	DELETE	DROP	TRUNCATE
Purpose	Remove rows	Remove table/database	Remove all rows
Structure	Keeps table	Removes table	Keeps table
WHERE clause	Yes	No	No
Rollback	Yes	No	No (usually)
Speed	Slower	Fast	Fastest
Auto-increment	Continues	N/A	Resets

sql

```sql
-- DELETE (conditional)
DELETE FROM employees WHERE id = 1;

-- TRUNCATE (all rows)
TRUNCATE TABLE employees;

-- DROP (entire table)
DROP TABLE employees;
```

13. INNER JOIN

Returns records that have matching values in both tables.

```sql
SELECT e.name, d.dept_name
FROM employees e
INNER JOIN departments d ON e.dept_id = d.dept_id;

-- Multiple joins
SELECT e.name, d.dept_name, p.project_name
FROM employees e
INNER JOIN departments d ON e.dept_id = d.dept_id
INNER JOIN projects p ON e.emp_id = p.emp_id;
```

14. LEFT and RIGHT JOIN

LEFT JOIN

Returns all records from left table, matching from right table.

RIGHT JOIN

Returns all records from right table, matching from left table.

```sql
-- LEFT JOIN
SELECT e.name, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id;

-- RIGHT JOIN
SELECT e.name, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.dept_id;

-- FULL OUTER JOIN (MySQL doesn't support, use UNION)
SELECT e.name, d.dept_name FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id
UNION
SELECT e.name, d.dept_name FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.dept_id;
```

15. Views

Virtual tables based on SQL queries.

```sql
```

```sql
-- Create view
CREATE VIEW high_salary_employees AS
SELECT name, salary, department
FROM employees
WHERE salary > 70000;

-- Use view
SELECT * FROM high_salary_employees;

-- Update view
CREATE OR REPLACE VIEW high_salary_employees AS
SELECT name, salary, department, hire_date
FROM employees
WHERE salary > 75000;

-- Drop view
DROP VIEW high_salary_employees;
```

16. Stored Procedures

Reusable SQL code blocks.

```sql
```

```sql
-- Create stored procedure
DELIMITER //
CREATE PROCEDURE GetEmployeesByDept(IN dept_name VARCHAR(50))
BEGIN
    SELECT * FROM employees
    WHERE department = dept_name;
END //
DELIMITER ;

-- Call procedure
CALL GetEmployeesByDept('IT');

-- Procedure with OUT parameter
DELIMITER //
CREATE PROCEDURE GetEmployeeCount(OUT emp_count INT)
BEGIN
    SELECT COUNT(*) INTO emp_count FROM employees;
END //
DELIMITER ;

-- Call with OUT parameter
CALL GetEmployeeCount(@count);
SELECT @count;

-- Drop procedure
DROP PROCEDURE GetEmployeesByDept;
```

SQL Query Solutions

1. Find the second highest salary

sql

```sql
-- Method 1: Using LIMIT with OFFSET
SELECT salary FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET 1;

-- Method 2: Using subquery
SELECT MAX(salary) FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);

-- Method 3: Using DENSE_RANK()
SELECT salary FROM (
    SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) as rank_num
    FROM employees
) ranked WHERE rank_num = 2;
```

2. Find the nth highest salary (3rd highest)

sql

```sql
-- Method 1: Using LIMIT
SELECT salary FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET 2;

-- Method 2: Using DENSE_RANK()
SELECT salary FROM (
    SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) as rank_num
    FROM employees
) ranked WHERE rank_num = 3;

-- Method 3: Generic function
DELIMITER //
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
    DECLARE result INT DEFAULT NULL;
    SET N = N - 1;
    SELECT salary INTO result
    FROM employees ORDER BY salary DESC LIMIT 1 OFFSET N;
    RETURN result;
END //
DELIMITER ;

SELECT getNthHighestSalary(3);
```

3. Get employees with duplicate salaries

```sql
-- Show employees with duplicate salaries
SELECT e1.* FROM employees e1
INNER JOIN (
    SELECT salary FROM employees
    GROUP BY salary HAVING COUNT(*) > 1
) e2 ON e1.salary = e2.salary
ORDER BY e1.salary;

-- Alternative method
SELECT * FROM employees WHERE salary IN (
    SELECT salary FROM employees
    GROUP BY salary HAVING COUNT(*) > 1
);
```

4. Find employees who don't have a manager

```sql
SELECT * FROM employees
WHERE manager_id IS NULL;

-- If manager is in same table
SELECT * FROM employees
WHERE manager_id IS NULL OR manager_id NOT IN (
    SELECT emp_id FROM employees WHERE emp_id IS NOT NULL
);
```

5. Retrieve the top 5 highest salaries

```sql
-- Top 5 distinct salaries
SELECT DISTINCT salary FROM employees
ORDER BY salary DESC LIMIT 5;

-- Top 5 employees by salary
SELECT * FROM employees
ORDER BY salary DESC LIMIT 5;
```

6. Get the department with the highest average salary

```sql
```

```sql
SELECT department, AVG(salary) as avg_salary
FROM employees
GROUP BY department
ORDER BY avg_salary DESC
LIMIT 1;
```

7. Find employees who joined in the last 30 days

sql

```sql
SELECT * FROM employees
WHERE hire_date >= DATE_SUB(CURRENT_DATE, INTERVAL 30 DAY);

-- Or using NOW()
SELECT * FROM employees
WHERE hire_date >= DATE_SUB(NOW(), INTERVAL 30 DAY);
```

8. Find employees who earn more than their manager

sql

```sql
SELECT e.name, e.salary, m.name as manager_name, m.salary as manager_salary
FROM employees e
INNER JOIN employees m ON e.manager_id = m.emp_id
WHERE e.salary > m.salary;
```

9. Find duplicate rows in a table

sql

```sql
-- Find duplicate records based on multiple columns
SELECT name, email, COUNT(*)
FROM employees
GROUP BY name, email
HAVING COUNT(*) > 1;

-- Show all duplicate rows
SELECT e1.* FROM employees e1
INNER JOIN (
    SELECT name, email FROM employees
    GROUP BY name, email HAVING COUNT(*) > 1
) e2 ON e1.name = e2.name AND e1.email = e2.email;
```

10. Get the highest paid employee in each department

```sql
-- Using window function
SELECT * FROM (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) as rn
    FROM employees
) ranked WHERE rn = 1;

-- Using correlated subquery
SELECT * FROM employees e1
WHERE salary = (
    SELECT MAX(salary) FROM employees e2
    WHERE e1.department = e2.department
);
```

11. Get employee count in each department (including departments with 0 employees)

```sql
SELECT d.dept_name, COUNT(e.emp_id) as employee_count
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_id, d.dept_name
ORDER BY d.dept_name;
```

12. Find the 3rd most recent hire

```sql
SELECT * FROM employees
ORDER BY hire_date DESC
LIMIT 1 OFFSET 2;

-- Using DENSE_RANK for ties
SELECT * FROM (
    SELECT *, DENSE_RANK() OVER (ORDER BY hire_date DESC) as hire_rank
    FROM employees
) ranked WHERE hire_rank = 3;
```

13. Find employees whose salary is above average

```sql
```

```sql
SELECT * FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

14. Find employees with the same salary as another employee

```sql
SELECT * FROM employees e1
WHERE EXISTS (
    SELECT 1 FROM employees e2
    WHERE e1.salary = e2.salary AND e1.emp_id != e2.emp_id
);

-- Alternative
SELECT * FROM employees
WHERE salary IN (
    SELECT salary FROM employees
    GROUP BY salary HAVING COUNT(*) > 1
);
```

15. Show department-wise 2nd highest salary

```sql
SELECT department, salary as second_highest_salary FROM (
    SELECT department, salary,
        DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) as rn
    FROM employees
) ranked WHERE rn = 2;
```

16. Get employees whose name starts with 'A' and ends with 'N'

```sql
SELECT * FROM employees
WHERE name LIKE 'A%N';

-- Case insensitive
SELECT * FROM employees
WHERE LOWER(name) LIKE 'a%n';
```

17. Find the highest, lowest, and average salary in one query

```sql
```

```sql
SELECT
    MAX(salary) as highest_salary,
    MIN(salary) as lowest_salary,
    AVG(salary) as average_salary
FROM employees;
```

18. Find employees who joined in the same year

sql

```sql
SELECT YEAR(hire_date) as hire_year, COUNT(*) as employee_count
FROM employees
GROUP BY YEAR(hire_date)
HAVING COUNT(*) > 1;

-- Show actual employees
SELECT * FROM employees
WHERE YEAR(hire_date) IN (
    SELECT YEAR(hire_date) FROM employees
    GROUP BY YEAR(hire_date) HAVING COUNT(*) > 1
)
ORDER BY hire_date;
```

19. Find employees with odd-numbered IDs

sql

```sql
SELECT * FROM employees
WHERE emp_id % 2 = 1;

-- Alternative using MOD function
SELECT * FROM employees
WHERE MOD(emp_id, 2) = 1;
```

20. Find employees who are in multiple departments

sql

```sql
-- Assuming employee_departments junction table
SELECT emp_id, COUNT(dept_id) as dept_count
FROM employee_departments
GROUP BY emp_id
HAVING COUNT(dept_id) > 1;

-- Show employee details
SELECT e.* FROM employees e
INNER JOIN (
    SELECT emp_id FROM employee_departments
    GROUP BY emp_id HAVING COUNT(dept_id) > 1
) multi_dept ON e.emp_id = multi_dept.emp_id;
```