Manual de Comandos SQL para MySQL 8.0

Disciplina de Banco de Dados T02

Este manual serve como um guia rápido para os principais comandos SQL.

1. Introdução à SQL

SQL (Structured Query Language) é a linguagem padrão para gerenciar e manipular bancos de dados relacionais. Ela é dividida em algumas subcategorias:

- **DDL (Data Definition Language):** Usada para definir, modificar e excluir a estrutura do banco de dados (esquemas, tabelas, etc.).
- **DML (Data Manipulation Language):** Usada para inserir, atualizar e excluir dados em tabelas.
- **DQL (Data Query Language):** Usada para consultar dados em tabelas.
- **DCL (Data Control Language):** Usada para controlar acessos e permissões.
- **TCL (Transaction Control Language):** Usada para gerenciar transações.

2. Criação, Alteração e Exclusão de Esquemas (DDL)

No MySQL, um "esquema" é sinônimo de "banco de dados".

2.1. Criação de Esquema/Banco de Dados

Para criar um novo banco de dados:

```
CREATE DATABASE nome_do_banco_de_dados;
```

Exemplo:

```
CREATE DATABASE minhaprimeiradb;
```

2.2. Uso de um Banco de Dados

Para selecionar um banco de dados para trabalhar:

```
USE nome_do_banco_de_dados;
```

Exemplo:

```
USE minhaprimeiradb;
```

2.3. Criação de Tabela

Após selecionar o banco de dados, você pode criar tabelas dentro dele:

```
CREATE TABLE nome_da_tabela (
    coluna1 tipo_de_dado [restrições],
    coluna2 tipo_de_dado [restrições],
    ...
    PRIMARY KEY (coluna_chave_primaria)
);
```

Exemplo:

```
CREATE TABLE alunos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    idade INT,
    curso VARCHAR(50)
);
```

Principais tipos de dados em MySQL:

- `INT`: Números inteiros.
- `VARCHAR(tamanho)`: Cadeia de caracteres de tamanho variável.
- `TEXT`: Cadeia de caracteres longas.
- `DATE`: Data (AAAA-MM-DD).
- `DATETIME`: Data e hora (AAAA-MM-DD HH:MM:SS).
- `DECIMAL(p, s)`: Números decimais com `p` dígitos no total e `s` dígitos após a vírgula.
- `BOOLEAN` / `TINYINT(1)`: Valores booleanos (MySQL armazena como 0 ou 1).

Principais Restrições (Constraints):

- `PRIMARY KEY`: Define uma coluna como chave primária (valores únicos e não nulos).
- `NOT NULL`: Garante que a coluna não possa ter valores nulos.
- `UNIQUE`: Garante que todos os valores em uma coluna sejam diferentes.
- `DEFAULT valor`: Define um valor padrão para a coluna se nenhum for especificado.
- `AUTO_INCREMENT`: Usado com chaves primárias numéricas para gerar valores automaticamente.
- `FOREIGN KEY (coluna) REFERENCES outra_tabela(outra_coluna)`: Define uma chave estrangeira para ligar tabelas.

2.4. Alteração de Tabela

Para adicionar, modificar ou excluir colunas de uma tabela:

Adicionar coluna:

```
ALTER TABLE nome_da_tabela
ADD COLUMN nova_coluna tipo_de_dado [restrições];
```

Exemplo:

```
ALTER TABLE alunos
ADD COLUMN email VARCHAR(100) UNIQUE;
```

Modificar coluna:

```
ALTER TABLE nome_da_tabela
MODIFY COLUMN nome_da_coluna novo_tipo_de_dado [novas_restrições];
```

Exemplo:

```
ALTER TABLE alunos
MODIFY COLUMN nome VARCHAR(150) NOT NULL;
```

Excluir coluna:

```
ALTER TABLE nome_da_tabela
DROP COLUMN nome_da_coluna;
```

Exemplo:

```
ALTER TABLE alunos
DROP COLUMN idade;
```

2.5. Exclusão de Tabela

Para excluir uma tabela inteira:

```
DROP TABLE nome_da_tabela;
```

Exemplo:

```
DROP TABLE alunos;
```

2.6. Exclusão de Esquema/Banco de Dados

Para excluir um banco de dados inteiro (cuidado, é irreversível!):

```
DROP DATABASE nome_do_banco_de_dados;
```

Exemplo:

```
DROP DATABASE minhaprimeiradb;
```

3. Inclusão, Alteração e Exclusão de Dados (DML)

Esses comandos manipulam os dados armazenados nas tabelas.

3.1. Inclusão de Dados (INSERT)

Para adicionar novas linhas (registros) a uma tabela:

```
INSERT INTO nome_da_tabela (coluna1, coluna2, ...)
VALUES (valor1, valor2, ...);
```

Você pode omitir a lista de colunas se estiver inserindo valores para todas as colunas na ordem em que foram definidas:

```
INSERT INTO nome_da_tabela
VALUES (valor1, valor2, ...);
```

Exemplo:

```
INSERT INTO alunos (nome, idade, curso)
VALUES ('João Silva', 20, 'Engenharia');
```

```
INSERT INTO alunos
VALUES (2, 'Maria Oliveira', 22, 'Ciência da Computação'); -- Se id for
AUTO_INCREMENT, não precisa especificar
```

3.2. Alteração de Dados (UPDATE)

Para modificar dados existentes em uma ou mais linhas de uma tabela:

```
UPDATE nome_da_tabela
SET coluna1 = novo_valor1, coluna2 = novo_valor2, ...
WHERE condicao; -- A cláusula WHERE é crucial para atualizar apenas as linhas
desejadas
```

Exemplo:

```
UPDATE alunos
SET curso = 'Sistemas de Informação'
WHERE id = 1;
```

```
UPDATE alunos
SET idade = 23, curso = 'Engenharia de Software'
WHERE nome = 'Maria Oliveira';
```

Cuidado: Se você omitir a cláusula `WHERE`, todos os registros da tabela serão atualizados.

3.3. Exclusão de Dados (DELETE)

Para remover uma ou mais linhas de uma tabela:

```
DELETE FROM nome_da_tabela
WHERE condicao; -- A cláusula WHERE é crucial para excluir apenas as linhas
desejadas
```

Exemplo:

```
DELETE FROM alunos
WHERE id = 2;
```

```
DELETE FROM alunos
WHERE curso = 'Engenharia';
```

Cuidado: Se você omitir a cláusula `WHERE`, todos os registros da tabela serão excluídos.

4. Consultas (DQL - SELECT)

A cláusula `SELECT` é a mais utilizada em SQL e permite recuperar dados de uma ou mais tabelas.

4.1. Selecionar Todas as Colunas

```
SELECT * FROM nome_da_tabela;
```

Exemplo:

```
SELECT * FROM alunos;
```

4.2. Selecionar Colunas Específicas

```
SELECT coluna1, coluna2, ... FROM nome_da_tabela;
```

Exemplo:

```
SELECT nome, curso FROM alunos;
```

4.3. Renomear Colunas (Aliases)

```
SELECT coluna AS novo_nome_coluna FROM nome_da_tabela;
```

Exemplo:

```
SELECT nome AS 'Nome do Aluno', curso AS Area FROM alunos;
```

4.4. Filtragem de Dados (WHERE)

Para selecionar linhas que satisfazem uma condição específica:

```
SELECT coluna1, coluna2 FROM nome_da_tabela
WHERE condicao;
```

Operadores comuns para `WHERE`:

- `=` (igual a)
- `<` ou `!=` (diferente de)
- `>` (maior que)
- `<` (menor que)
- `>=` (maior ou igual a)
- `<=` (menor ou igual a)

- `BETWEEN valor1 AND valor2` (entre dois valores, inclusivo)
- `LIKE padrao` (busca por padrões em strings)
- `IN (valor1, valor2, ...)` (verifica se um valor está em uma lista)
- `IS NULL` (verifica se o valor é nulo)
- `IS NOT NULL` (verifica se o valor não é nulo)
- `AND`, `OR`, `NOT` (operadores lógicos para combinar condições)

Exemplos:

```
SELECT *FROM alunos WHERE idade > 20;
SELECT nome, curso FROM alunos WHERE curso = 'Engenharia' AND idade < 25;
SELECT* FROM alunos WHERE nome LIKE 'J%'; -- Nomes que começam com 'J'
SELECT *FROM alunos WHERE curso IN ('Engenharia', 'Direito');
SELECT* FROM alunos WHERE email IS NULL;
```

4.5. Ordenação de Resultados (ORDER BY)

Para ordenar os resultados em ordem crescente (`ASC`) ou decrescente (`DESC`):

```
SELECT coluna1, coluna2 FROM nome_da_tabela
ORDER BY coluna_para_ordenar ASC/DESC;
```

Exemplo:

```
SELECT nome, idade FROM alunos ORDER BY idade DESC;
SELECT nome, curso FROM alunos ORDER BY curso ASC, nome ASC; -- Ordena por curso,
e dentro de cada curso, por nome
```

4.6. Limitar Resultados (LIMIT)

Para retornar um número específico de linhas:

```
SELECT * FROM nome_da_tabela
LIMIT numero_de_linhas;
```

Exemplo:

```
SELECT * FROM alunos LIMIT 5; -- Retorna as primeiras 5 linhas
```

Você pode usar `LIMIT` com um offset para pular um número de linhas antes de começar a retornar:

```
SELECT * FROM nome_da_tabela
LIMIT offset, numero_de_linhas;
```

Exemplo:

```
SELECT * FROM alunos LIMIT 5, 10; -- Pula as primeiras 5 linhas e retorna as
próximas 10
```

4.7. Junções (JOIN)

Para combinar linhas de duas ou mais tabelas com base em uma coluna relacionada entre elas.

- `INNER JOIN`: Retorna linhas quando há correspondência em ambas as tabelas.

```
SELECT *
FROM tabela1
INNER JOIN tabela2 ON tabela1.coluna_comum = tabela2.coluna_comum;
```

Exemplo: Suponha uma tabela `cursos` (`id_curso`, `nome_curso`) e `alunos` (`id`, `nome`, `id_curso`).

```
SELECT alunos.nome, cursos.nome_curso
FROM alunos
INNER JOIN cursos ON alunos.id_curso = cursos.id_curso;
```

- `LEFT JOIN` **(ou** `LEFT OUTER JOIN`**)**: Retorna todas as linhas da tabela da esquerda e as linhas correspondentes da tabela da direita. Se não houver correspondência, o resultado da tabela da direita terá `NULL`.

```
SELECT *
FROM tabela1
LEFT JOIN tabela2 ON tabela1.coluna_comum = tabela2.coluna_comum;
```

- `RIGHT JOIN` **(ou** `RIGHT OUTER JOIN`**)**: Retorna todas as linhas da tabela da direita e as linhas correspondentes da tabela da esquerda. Se não houver correspondência, o resultado da tabela da esquerda terá `NULL`.

```
SELECT *
FROM tabela1
RIGHT JOIN tabela2 ON tabela1.coluna_comum = tabela2.coluna_comum;
```

- `FULL JOIN` **(ou** `FULL OUTER JOIN`**)**: Retorna todas as linhas quando há uma correspondência em uma das tabelas. (MySQL não suporta `FULL JOIN` diretamente, mas pode ser simulado com `UNION` de `LEFT JOIN` e `RIGHT JOIN`).

5. Operações de Conjunto (SET Operations)

Essas operações combinam os resultados de duas ou mais consultas `SELECT`. As colunas selecionadas em cada `SELECT` devem ter o mesmo número, a mesma ordem e tipos de dados compatíveis.

- `UNION`: Retorna todas as linhas distintas de ambas as consultas.

```
SELECT coluna1, coluna2 FROM tabela1
UNION
SELECT coluna1, coluna2 FROM tabela2;
```

- `UNION ALL`: Retorna todas as linhas de ambas as consultas, incluindo duplicatas.

```
SELECT coluna1, coluna2 FROM tabela1
UNION ALL
SELECT coluna1, coluna2 FROM tabela2;
```

Exemplo: Suponha que você tenha duas tabelas, `alunos_atuais` e `alunos_formados`, com colunas `nome` e `email`.

```
UNION
SELECT nome, email FROM alunos_formados; -- Retorna todos os alunos únicos de
ambas as tabelas
```

- `INTERSECT`: Retorna apenas as linhas que aparecem em ambas as consultas. (MySQL não suporta `INTERSECT` diretamente, mas pode ser simulado com `INNER JOIN` ou `IN`).

  Simulação com `INNER JOIN`**:**

```
SELECT t1.coluna1, t1.coluna2
FROM tabela1 t1
INNER JOIN tabela2 t2 ON t1.coluna1 = t2.coluna1 AND t1.coluna2 = t2.coluna2;
```

- `EXCEPT` **(ou** `MINUS`**)**: Retorna as linhas que estão na primeira consulta, mas não na segunda. (MySQL não suporta `EXCEPT` diretamente, mas pode ser simulado com `LEFT JOIN` e `WHERE IS NULL`).

  Simulação com `LEFT JOIN`**:**

```
SELECT t1.coluna1, t1.coluna2
FROM tabela1 t1
```

```
    LEFT JOIN tabela2 t2 ON t1.coluna1 = t2.coluna1 AND t1.coluna2 = t2.coluna2
    WHERE t2.coluna1 IS NULL;
```

6. Funções Agregadas

Funções agregadas realizam cálculos em um conjunto de linhas e retornam um único valor.

- `COUNT()`: Conta o número de linhas.

```
    SELECT COUNT(*) FROM alunos; -- Conta todas as linhas
    SELECT COUNT(idade) FROM alunos; -- Conta linhas onde 'idade' não é NULL
    SELECT COUNT(DISTINCT curso) FROM alunos; -- Conta cursos únicos
```

- `SUM()`: Calcula a soma dos valores de uma coluna numérica.

```
    SELECT SUM(idade) FROM alunos;
```

- `AVG()`: Calcula a média dos valores de uma coluna numérica.

```
    SELECT AVG(idade) FROM alunos;
```

- `MIN()`: Retorna o menor valor de uma coluna.

```
    SELECT MIN(idade) FROM alunos;
    SELECT MIN(nome) FROM alunos; -- Funciona também com strings (ordem
alfabética)
```

- `MAX()`: Retorna o maior valor de uma coluna.

```
    SELECT MAX(idade) FROM alunos;
    SELECT MAX(nome) FROM alunos;
```

7. Agregação com Agrupamento (GROUP BY)

A cláusula `GROUP BY` é usada em conjunto com funções agregadas para agrupar linhas que têm os mesmos valores em colunas especificadas. As funções agregadas então operam em cada grupo individualmente.

```
SELECT coluna_agrupadora, FUNCAO_AGREGADA(coluna)
FROM nome_da_tabela
GROUP BY coluna_agrupadora;
```

Exemplos:

```sql
SELECT curso, COUNT(*) AS total_alunos
FROM alunos
GROUP BY curso; -- Conta quantos alunos há em cada curso
```sql

```sql
SELECT curso, AVG(idade) AS media_idade
FROM alunos
GROUP BY curso; -- Calcula a média de idade por curso
```

```sql
SELECT curso, SUM(CASE WHEN idade < 20 THEN 1 ELSE 0 END) AS alunos_menores_20
FROM alunos
GROUP BY curso; -- Conta alunos menores de 20 por curso
```

7.1. Filtragem de Grupos (HAVING)

A cláusula `HAVING` é usada para filtrar grupos criados pela cláusula `GROUP BY`, de forma similar ao `WHERE` que filtra linhas individuais.

```sql
SELECT coluna_agrupadora, FUNCAO_AGREGADA(coluna)
FROM nome_da_tabela
GROUP BY coluna_agrupadora
HAVING condicao_do_grupo;
```

Exemplo:

```sql
SELECT curso, COUNT(*) AS total_alunos
FROM alunos
GROUP BY curso
HAVING COUNT(*) > 2; -- Mostra apenas os cursos que têm mais de 2 alunos
```

```sql
SELECT curso, AVG(idade) AS media_idade
FROM alunos
GROUP BY curso
HAVING AVG(idade) > 21; -- Mostra cursos onde a média de idade é maior que 21
```

8. Subconsultas Aninhadas (Subqueries)

Uma subconsulta (ou subquery) é uma consulta `SELECT` dentro de outra consulta SQL. Elas podem ser usadas em diversas partes de uma instrução SQL, como `WHERE`, `FROM`, `SELECT` ou `HAVING`.

8.1. Subconsultas na Cláusula `WHERE`

Comumente usadas com operadores como `IN`, `NOT IN`, `=`, `>`, `<`, `>=`, `<=`, `EXISTS`, `NOT EXISTS`.

Exemplo com `IN`:

```
-- Selecionar alunos que estão em cursos com mais de 30 alunos
SELECT nome, curso
FROM alunos
WHERE curso IN (SELECT curso FROM alunos GROUP BY curso HAVING COUNT(*) > 30);
```

Exemplo com operador de comparação:

```
-- Selecionar alunos com idade acima da média
SELECT nome, idade
FROM alunos
WHERE idade > (SELECT AVG(idade) FROM alunos);
```

Exemplo com `EXISTS`:

```
-- Selecionar cursos que têm pelo menos um aluno
SELECT nome_curso
FROM cursos
WHERE EXISTS (SELECT 1 FROM alunos WHERE alunos.id_curso = cursos.id_curso);
```

8.2. Subconsultas na Cláusula `FROM` (Tabelas Derivadas)

Uma subconsulta na cláusula `FROM` atua como uma tabela temporária. É necessário dar um alias a essa "tabela".

```
SELECT t1.coluna, t2.coluna
FROM tabela1 t1
INNER JOIN (SELECT coluna_sub, outra_coluna FROM tabela_interna WHERE condicao) AS
t2
ON t1.coluna = t2.coluna_sub;
```

Exemplo:

```
-- Encontrar o nome do aluno e a média de idade do seu curso
SELECT a.nome, c.media_idade_curso
FROM alunos a
INNER JOIN (
    SELECT curso, AVG(idade) AS media_idade_curso
    FROM alunos
```

```
    GROUP BY curso
) AS c ON a.curso = c.curso;
```

8.3. Subconsultas na Cláusula `SELECT` (Subconsultas Correlacionadas)

Uma subconsulta na cláusula `SELECT` é executada para cada linha do conjunto de resultados da consulta externa.

SELECT coluna_principal, (SELECT COUNT(*) FROM tabela_relacionada WHERE condicao_correlacionada) AS contagem FROM tabela_principal;

Exemplo:

```
-- Listar o nome de cada curso e o número total de alunos nesse curso
SELECT
    c.nome_curso,
    (SELECT COUNT(*) FROM alunos a WHERE a.id_curso = c.id_curso) AS total_alunos
FROM
    cursos c;```
```