T03: SQL + MySQL

1. SQL

1. SQL
    1. *Data Definition Language*
    2. Domínios em SQL
    3. Construção de Tabela
        3.1. Sintaxe
        3.2. Restrições de Integridade
    4. `DROP` e `ALTER` de Tabela
    5. *Queries*
        5.1. Estrutura Típica
        5.2. Clausula `SELECT`
        5.3. Clausula `WHERE`
        5.4. Clausula `FROM`
        5.5. Operação `RENAME`
        5.6. Operações em *Strings*
        5.7. Clausula `ORDER BY`
        5.8. Operações sobre Conjuntos
        5.9. Funções Agregadoras
        5.9 Função Agregadora/Clausula `GROUP BY`
        5.10. Função Agregadora/Clausula `HAVING`
        5.11. Valores `NULL`
        5.12. Subqueries Aninhadas
        5.13. Clausula `WITH`
    6. Views
        6.1. Definição
    7. Eliminação de Valores
    8. Inserção de Valores
    9. *Update* de Valores
    10. *Join* em Relações
        10.1. Condição
        10.2. Tipos
2. SQL Avançado e MySQL
    2.1. Tipos de Dados Embutidos no SQL
        2.1.1. Extração e *Casting*
    3. Tipos de Dados Definidos
    4. Restrições de Domínio
    5. Tipos de Dados Grandes (***Large-Object*, LO**)
    6. Restrições de Integridade

6.1. Restrições numa Relação
7. Integridade Referencial
8.1. Formas de Alteração de *Schema*
8.2. Especificação de Autorização
9. Privilégios em SQL
10. Revogação de Privilégios
11. Triggers

1. *Data Definition Language*

- Permite a especificação de um conjunto de relações, bem como informação relativa às mesmas:
  - Esquema para cada relação;
  - O dominio de valores associado a cada atributo;
  - Restrições de integridade;

2. Domínios em SQL

- `CHAR(n)` : *strings* de tamanho fixo `n`;
- `VARCHAR(n)` : *strings* de tamanho variável, com tamanho máximo de `n` ;
- `INT`
- `SMALLINT`
- `NUMERIC(p,d)`: número com casas fixas, onde:
  - `p`: precisão
  - `d`: número de digitos decimais
- `REAL`, `DOUBLE PRECISION`: número de vírgula flutuante e número de vírgula flutuante com dupla precisão;
- `FLOAT(n)`: número de vírgula flutuante;

3. Construção de Tabela

3.1. Sintaxe

```
CREATE TABLE R(A1 D1, A2 D2, ..., An Dn,
              (restricao-integridade-1),
```

```
            ...,
            (restricao-integridade-k))
```

- R: nome da relação;
- A_i: nome de atributo;
- D_i: tipo de dados do atributo A_i.

3.2. Restrições de Integridade

- `NOT NULL`
- `PRIMARY KEY(A1,...,An)`

> ❗ `PRIMARY_KEY` atribui automaticamente a RI `NOT NULL` ao atributo.

4. `DROP` e `ALTER` de Tabela

- `DROP`: apaga toda a informação da relação na base de dados;
- `ALTER`: utilizado para adicionar atributos (1) ou remover atributos (2) de uma relação existente:

```
ALTER TABLE R ADD A D -- (1)
ALTER TABLE R DROP A -- (2)
```

  - No caso de `(1)`, todos os tuplos da relação são atribuídos o valor `null` para o novo atributo;
  - Nem todas as bases de dados suportam `(2)`.

5. *Queries*

5.1. Estrutura Típica

```
SELECT A1, A2,..., An
FROM R1, R2,..., RM
WHERE P
```

- A_i: nome de atributo;

- R: nome da relação;
- P: predicado;
- **Resulta numa relação.**

É representado em álgebra *booleana* pela expressão

$$\prod_{A1,\ldots,A_n} (\sigma_p(r_1 \times \cdots \times r_m))$$

> Projection (relational algebra) - Wikipedia
>
> In relational algebra, a projection is a unary operation written as , where is a relation and are attribute names. Its result is defined as the set obtained when the components of the tuples in are restricted to the set - it discards (or excludes) the other attributes.[1] In practical terms, if a relation is thought of as a table, then projection can be thought of as
>
> 𝕎 https://en.wikipedia.org/wiki/Projection_(relational_algebra)

> Selection (relational algebra) - Wikipedia
>
> In relational algebra, a selection (sometimes called a restriction in reference to E.F. Codd's 1970 paper and not, contrary to a popular belief, to avoid confusion with SQL's use of SELECT, since Codd's article predates the existence of SQL) is a unary operation that denotes a subset of a relation.
>
> 𝕎 https://en.wikipedia.org/wiki/Selection_(relational_algebra)

> Rename (relational algebra) - Wikipedia
>
> From Wikipedia, the free encyclopedia In relational algebra, a rename is a unary operation written as where: The result is identical to R except that the b attribute in all tuples is renamed to a.
>
> 𝕎 https://en.wikipedia.org/wiki/Rename_(relational_algebra)

5.2. Clausula `SELECT`

- Lista atributos que se quer retornar no resultado da *query*;
- É representado em álgebra *booleana* pela expressão

$$\prod_{A}(R)$$

- Os nomes em SQL são *case-insensitive*;
- SQL permite valores duplicados nas relações:
  - Forçar a eliminação com a keyword `DISTINCT`, e.g.

```
SELECT DISTINCT A
FROM R
```

- `*` na clausula significa "todos os atributos", e.g.

```
SELECT *
FROM R
```

- A clausula `SELECT` pode ter operadores aritméticos:

```
SELECT A1, A2, A3 * 100
FROm R
```

5.3. Clausula `WHERE`

- Específica as condições que o resultado deve satisfazer;
- A comparação de resultados pode ser combinado com conetivo lógicos `AND`, `OR` , `NOT` e `BETWEEN` (inclusívo);
- Comparações podem ser aplicadas a resultados de expressões aritméticas, e.g.
  -

```
SELECT A
FROM R
WHERE B BETWEEN 5 AND 10
```

5.4. Clausula `FROM`

- Lista as relações involvidas na *query*, e.g.

```
SELECT A, R1.B, C
FROM R1, R2
WHERE R1.B = R2.B AND D = "Lorem"
```

5.5. Operação `RENAME`

- SQL permite renomear atributos com a clausula `AS`, e.g:

```
SELECT A, B as X, C
FROM R1, R2
WHERE R1.B = R2.B
```

5.6. Operações em *Strings*

- O operador `LIKE` usa os seguintes *patterns*:
  - `%`: corresponde a qualquer *substring*;
  - `_`: corresponde a qualquer carater.
- SQL suporta várias operações, tais como:
  - Concatenação: `||` ;
  - Conversão entre minúsculas e maísculas;
  - Tamanho de *string* , extração de *substrings*, *etc.*

5.7. Clausula `ORDER BY`

- `DESC` para ordem descendente;
- `ASC` para ordem ascendente;

```
SELECT DISTINCT A
FROM R1, R2
WHERE R1.B = R2.B AND C = "Lorem"
ORDER BY A DESC -- ou ORDER BY A ASC
```

5.8. Operações sobre Conjuntos

- As operações **união, interseção** e **excepto** representam-se por $\cup, \cap$ e $\setminus$, respetivamente;
- As operações eliminam valores duplicados:
  - Para manter duplicados, usar `UNION ALL`, `INTERSECT ALL` e `EXCEPT ALL`.
- Supondo que um tuplo ocorre m vezes em r e n vezes em s, então ocorre:
  - $m + n$ vezes em `R UNION ALL S`;
  - $min(m, n)$ vezes em `R INTERSECT ALL S`;
  - $max(0, m - n)$ vezes em `R EXCEPT ALL S.`

- **EXEMPLOS**:
  - Encontrar todos os *customers* que têm um *loan*, uma *account*, ou ambos:

```
SELECT customer_name FROM depositor
UNION
SELECT customer_name FROM borrower
```

  - Encontrar todos os *customers* que, simultaneamente, um *loan e* uma *account:*

```
SELECT customer_name FROM depositor
INTERSECT
SELECT customer_name FROM borrower
```

  - Encontrar todos os *customers* que têm uma *account* mas não um *loan:*

```
SELECT customer_name FROM depositor
EXCEPT
SELECT customer_name FROM borrower
```

5.9. Funções Agregadoras

- `AVG`: valor médio;

```
SELECT AVG (balance)
FROM account
WHERE branch_name = 'Perryridge';
```

- `MIN`: valor mínimo;
- `MAX`: valor máximo;
- `SUM`: soma de valores;
- `COUNT` : número de valores;

```
SELECT COUNT(*)
FROM CUSTOMER
```

```
SELECT COUNT(DISTINCT customer_name)
FROM CUSTOMER
```

5.9 Função Agregadora/Clausula `GROUP BY`

- Agrupa tuplos que têm os mesmos valores em tuplos "sumários", e.g. "encontrar o número de clientes in cada país".
    - Os atributos na clausula `SELECT` (à exceção daqueles em funções agregadoras) têm de aparecer na lista `GROUP BY:`

```
SELECT branch_name, COUNT(DISTINCT customer_name)
FROM depositor, account
WHERE depositor.account_number = account.account_number
GROUP BY branch_name
```

5.10. Função Agregadora/Clausula `HAVING`

- Tem o mesmo uso da clausula `WHERE`, mas esta última não pode ser utilizada com funções agregadoras:

```
SELECT branch_name, AVG(balance)
FROM ACCOUNT
GROUP BY branch_name
HAVING AVG(balance) > 1200
```

5.11. Valores `NULL`

- Significa um valor desconhecido ou que não existe;
- O predicado `IS NULL` pode ser utilizado para verificar se valores não existem;
- O valor de uma expressão aritmética que envolve `NULL`, é também `NULL`:

```
5 + NULL -- NULL
```

- Funções agregadoras ignoram `NULL`:
    - Qualquer comparação com `NULL` retorna `NULL`:
    - Logica triplamente-valorada utilizando o valor de verdade `UNKNOWN`:
        - `OR`:

```
UNKNOWN OR TRUE -- TRUE
UNKNOWN OR FALSE -- FALSE
```

```
UNKNOWN OR UNKNOWN -- UNKNOWN
```

- `AND`:

```
TRUE AND UNKNOWN -- UNKNOWN
FALSE AND UNKNOWN -- FALSE
UNKNOWN AND UNKNOWN -- UNKNOWN
```

- `NOT`:

```
NOT UNKNOWN -- UNKNOWN
```

- Todas as funções agregadoras, à exceção de `COUNT(*)`, ignoram tuplos com valores `NULL` nos atributos agregados.

5.12. Subqueries Aninhadas

- Uma *subquery* é uma expressão do tipo `SELECT FROM ... WHERE ...` que está aninhada em outra query;
- As *subqueries* são utilizadas para verificar se um tuplo pertence a um conjunto, para comparação de conjuntos e para a cardinalidade de conjuntos;

```
SELECT DISTINCT customer_name
FROM borrower
WHERE customer_name IN (SELECT customer_name
                        FROM depositor)
```

5.13. Clausula `WITH`

- Providencía uma maneira de definir uma *view* temporária cuja definição está apenas disponível para a query na qual a clausula aparece.

```
WITH max_balance (value) AS
 SELECT max (balance)
 FROM account
SELECT account_number
FROM account, max_balance
WHERE account.balance = max_balance.value
```

6. Views

- Em alguns casos, pode não ser desejável que todos os utilizadores vejam todo o modelo lógico da BD;
- Uma *view* providencia um mecanismo para esconder certos dados da vista de alguns utilizadores;
- Qualquer relação que não esteja no modelo conceptual mas é tornada visível a um utilizador como uma "relação virtual" é uma *view;*

6.1. Definição

- Criação:

```
CREATE VIEW v AS <query>
```

  - v: nome da vista;
  - $query$: qualquer expressão legal em SQL.

- Exemplos:

```
CREATE VIEW all_customers AS
  (
    SELECT branch_name, customer_name
    FROM depositor, account
    WHERE depositor.account_number = account.account_number)
  )
  UNION
  (
    SELECT branch_name, customer_name
    FROM borrower, loan
    WHERE borrower.loan_number = loan.loan_number
  )

--- Query
SELECT customer_name
FROM all_customer
WHERE branch_name = 'Perryridge'
```

7. Eliminação de Valores

```
DELETE FROM account
WHERE branch_name = 'Perryridge'
```

8. Inserção de Valores

```
INSERT INTO account
VALUES ('A-9732', 'Perryridge', 1200)

-- ou

INSERT INTO account(branch_name, balance, account_number)
VALUES ('Perryridge', 1200, 'A-9732')
```

9. *Update* de Valores

```
UPDATE account
SET balance = balance * 1.06
WHERE balance > 10000

UPDATE account
SET balance = balance * 1.05
WHERE balance <= 10000

-- ou

UPDATE account
SET balance = CASE
  WHEN balance <= 10000 then balance * 1.05
  ELSE balance * 1.06
END
```

10. *Join* em Relações

- Com *joins*, é possíevl retornar dados de duas ou mais relações com base em relacionamentos lógicos entre essas mesmas relações;

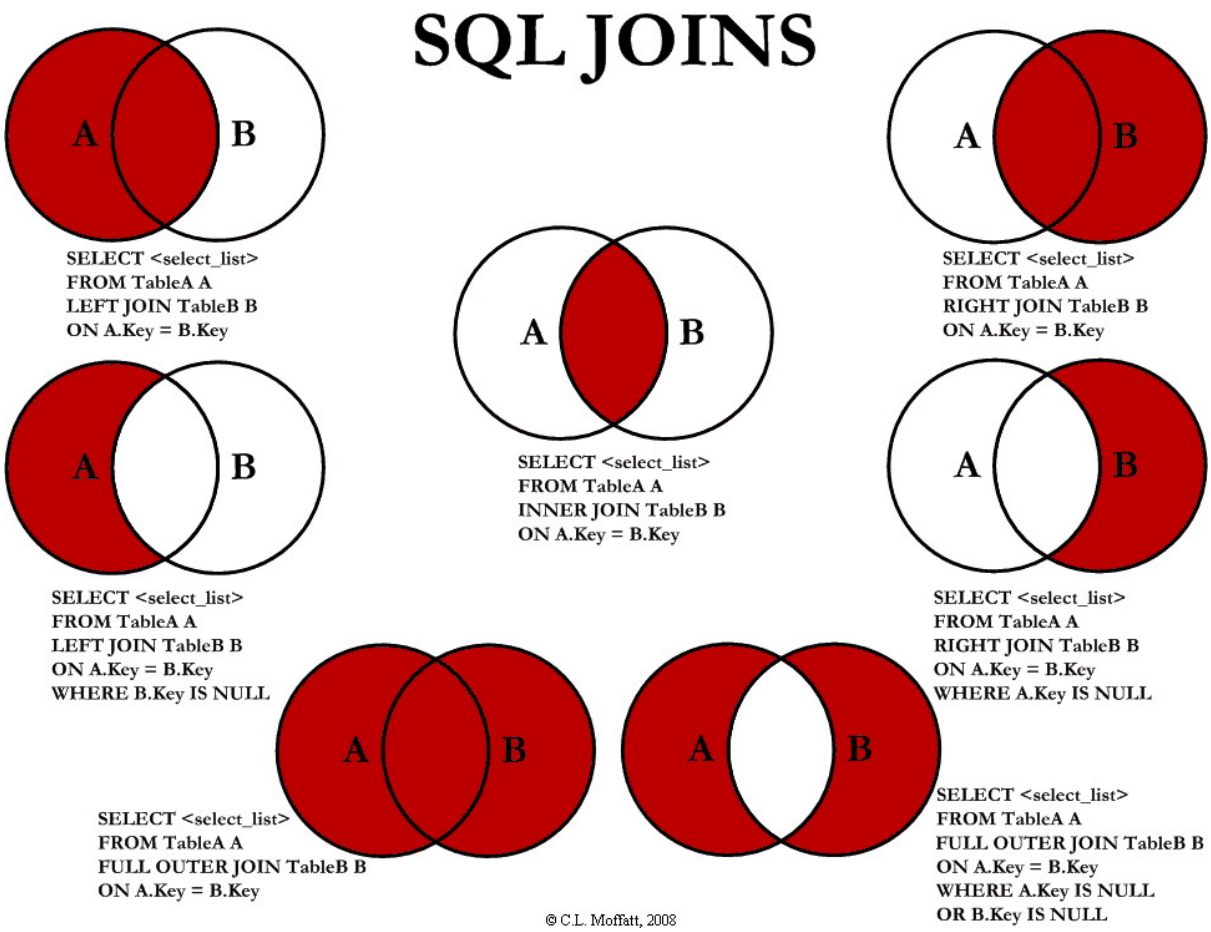
10.1. Condição

- A condição de *join* de fine a maneira como duas relações estão relacionadas numa query ao:
  - Especificar a coluna de cada tabela utilizada para o *join;*
  - Especificar um operador lógico, utiliado para comparar valores das colunas acima referidas;

10.2. Tipos

- **Inner**: `A INNER B` retorna o resultado de $A \cap B$;

- Outer: `A OUTER B` retorna o resultado de $A \cup B$;
- `INNER JOIN`
  - Retorna todos os tuplos de ambas as relações desde que o predicado seja satisfeito;
- `LEFT OUTER JOIN`
  - Retorna todos os tuplos da relação da esquerda e os tuplos da relação da direita que satisfazem o predicado;
  - Para os tuplos que não têm um tuplo correspondente na tabela da direita, o resultado será `NULL`;
- `RIGHT OUTER JOIN`
  - Retorna os tuplos da relação da esquerda que satisfazem o predicado e todos os tuplos da relação da direita;
  - Para os tuplos que não têm um tuplo correspondente na tabela da esquerda, o resultado será `NULL`;
- `FULL OUTER JOIN`
  - Retorna todos os tuplos de ambas as relações



---

2. SQL Avançado e MySQL

2.1. Tipos de Dados Embutidos no SQL

- `DATE`:
  - Datas, contendo um ano de 4 dígitos, mês e data do dia;

```
DATE '2005-7-27'
```

- `TIME`:
  - Hora do dia, em horas, minutos e segundos;

```
TIME '09:00:30'
TIME '09:00:30.75'
```

- `TIMESTAMP`:
  - `DATE` + `TIME`;

```
TIMESTAMP '2005-7-37 09:00:30.75'
```

- `INTERVAL`:
  - Período de tempo;
  - A subtração de um dos tipos acima resulta num `INTERVAL`;
  - `INTERVAL` pode ser adicionado a um dos tipos acima;

```
INTERVAL '1' DAY
```

2.1.1. Extração e *Casting*

- É possível extraír campos individuais dos tipos acima:

```
EXTRACT (YEAR FROM r.starttime)
```

- É possível fazer *cast* de *strings* para um dos tipos acima:

```
CAST <STRING> AS DATE
CAST <STRING> AS TIME
CAST <STRING> AS TIMESTAMP
```

3. Tipos de Dados Definidos

- `CREATE TYPE` permite criar um tipo de dados :

```
CREATE TYPE Dollars AS NUMERIC(12,2) FINAL
```

- `CREATE DOMAIN` permite também criar um tipo de dados:

```
CREATE DOMAIN person_name CHAR(20) NOT NULL
```

  - Similar a `TYPE`;

- Um `DOMAIN` pode ter restrições (e.g. `NOT NULL`).

4. Restrições de Domínio

- A forma mais elementar de implementar restrições de integridade;
- Novos `DOMAIN`s podem ser criados a partir de tipos de dados existentes:

```
CREATE DOMAIN Dollars NUMERIC(12,2)
CREATE DOMAIN Pounds NUMERIC(12,2)
```

  - Apesar de partilharem o mesmo tipo de dados, não é possível comparar ambos os valores;
  - É possível fazer *casting* e, posteriormente, comparar.

5. Tipos de Dados Grandes (*Large-Object*, LO)

- `BLOB`:
  - BINARY LARGE OBJECT;
  - Coleção de dados binários não interpretados:
    - A interpretação é feita por uma aplicação fora do SGBD;
- `CLOB`:
  - CHARACTER LARGE OBJECT;
  - Coleção de `chars`;
- Quando uma *query* retorna um LO, é retornado um *pointer*;

6. Restrições de Integridade

- Não permitem alterações acidentais à BD:
  - Asseguram que modificações autorizadas não resultam na perca de dados.

6.1. Restrições numa Relação

- `NOT NULL`;
- `PRIMARY KEY`;
- `UNIQUE`:

    - `UNIQUE (A1, A2, ..., Am)`
    - Especifica que os atributos `A1, A2, ..., Am` formam uma chave candidata (CK):
        - Estas podem ser `NULL`, ao contrário de PKs.
- `` `CHECK(P)` ``, onde P é um predicado.

7. Integridade Referencial

- Garante que um valor que aparece numa relação para um dado conjunto de atributos, também aparece para um dado conjunto de atributos noutra relação:
    - Se "Perryridge" é o nome de um banco que aparece num dos tuplos da relação `account`, então existe um tuplo na relação `branch` para "Perryridge".
- PKs, FKs e CKs podem ser especificadas em `CREATE TABLE`;
    - `PRIMARY KEY` lista as PKs,
    - `UNIQUE KEY` lista as CKs;
    - `FOREIGN KEY` lista as FKs;

8. Autorização

- `READ`
    - Permite apenas a leitura de dados;
- `INSERT`
    - Permite inserção de dados mas não a modificação de dados existentes;
- `UPDATE`
    - Permite modificação mas não eliminação de dados;
- `DELETE`:
    - Permite eliminação de dados;

8.1. Formas de Alteração de *Schema*

- `INDEX`
    - Permite a criação e eliminação de índices;
- `RESOURCES`
    - Permite a criação de novas relações;

- `ALTERATION`
  - Permite a adição ou eliminação de atributos numa relação;
- `DROP`
  - Permite eliminar relações;

8.2. Especificação de Autorização

```
GRANT <PRIVILEGE LIST>
ON <RELATION/VIEW NAME> TO <USER LIST>
```

- Conceder uma permissão numa *view* não implica conceder privilégios a relações subjacentes;

9. Privilégios em SQL

`SELECT`

- Permite acesso de leitura a uma relação ou habilidade de fazer uma *query* utilizando a view:

  ```
  GRANT SELECT ON branch TO U1, U2, U3
  ```

  onde U_n são utilizadores.

- `INSERT`
  - Permite inserir tuplos;
- `UPDATe`
  - Permite atualizar tuplos;
- `DELETE`
  - Permite eliminar tuplos;
- `ALL PRIVILEGES`

10. Revogação de Privilégios

- `REVOKE`

```
REVOKE <PRIVILEGE LIST>
ON <RELATION NAME or VIEW NAME> FROM <USER LIST>
```

11. Triggers

- É um comando que é executado automaticamente pelo sistema como um *side effect* de uma modificação na BD;
    - Especificar as condições sob as quais o trigger deve executar;
    - Especificar as ações a serem tomadas quando o trigger executa;
- Os eventos podem ser `INSERT`, `DELETE` ou `UPDATE`;