DDL – *DATA DEFINITION LANGUAGE*

CRIANDO ESQUEMA

O conceito de um esquema SQL foi definido com o SQL2 para agrupar tabelas e outros elementos que pertençam a mesma aplicação. Um esquema é identificado por um nome, um identificador do usuário ou conta que possui o esquema e descritores de cada elemento do esquema.

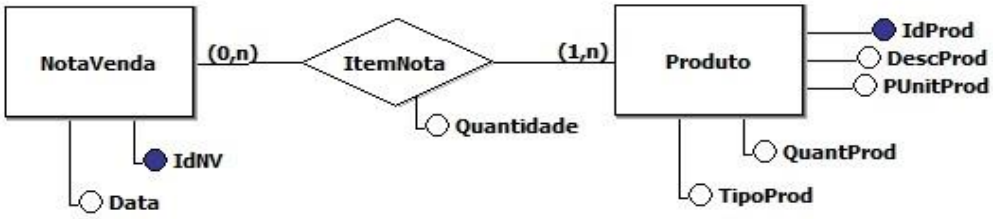
Sintaxe:

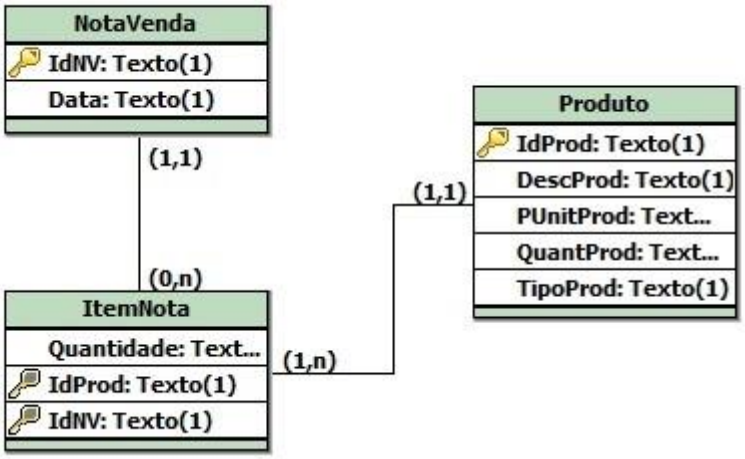
CREATE SCHEMA nome_schema;

Exemplo(s):

CREATE SCHEMA aula3;

MODELO DE EXEMPLO DER:





CRIANDO TABELAS

Sintaxe:

```
CREATE TABLE nome_tabela (
<definição_coluna>
[, <definição_coluna> | <restrição_tabela> ]);
<definição_coluna> = nome_coluna
{<tipo_dado> | COMPUTED [BY] (<expr>) | domínio}
[DEFAULT {literal | NULL | USER}]
[NOT NULL]
[<restrição_coluna>]
<restrição_coluna> = [CONSTRAINT nome_restrição]
{ UNIQUE
| PRIMARY KEY
| REFERENCES outra_tabela [(outra_coluna [,outra_coluna])]
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
| CHECK (<condição>)}
<restrição_tabela> = [CONSTRAINT nome_restrição]
{{PRIMARY KEY | UNIQUE} (coluna [, coluna])
| FOREIGN KEY (coluna [, coluna]) REFERENCES outra_tabela
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}] |
CHECK (<condição>)}
```

Exemplo(s):

- restrição_coluna: referencia somente uma coluna, aceitando todos os tipos de restrições.
- restrição_tabela: referencia uma ou mais colunas. Só não aceita o tipo NOT NULL.

Tipos de restrições

[NOT] NULL - Indica se a coluna pode ou não receber valores nulos. O default é NULL.

UNIQUE - Indica que a coluna ou combinação de colunas não pode ter valores repetidos.

PRIMARY KEY - Indica que a coluna ou combinação de colunas forma a chave primária.

Chave Estrangeira (REFERENCES ou FOREIGN KEY) - Usada a nível de coluna, indica que a coluna é uma chave estrangeira. Usada a nível de tabela, indica que a coluna ou combinação de colunas é uma chave estrangeira.

ON UPDATE | ON DELETE - indica a ação a ser tomada quando uma linha na tabela referenciada é atualizada ou removida.

NO ACTION - o valor da FK não é mudado. Pode causar erro na atualização/remoção da chave primária da tabela referenciada, devido a integridade referencial. É a opção *default*.

CASCADE - o valor da FK é alterado de acordo com a alteração feita na chave primária. No caso de remoção, as linhas correspondentes são removidas.

SET DEFAULT - o valor da FK recebe o valor definido com *default* para aquela coluna.

SET NULL - o valor da FK recebe NULL.

CHECK - Não permite que valores que violem a condição estabelecida sejam gravados na coluna.

Exemplo(s):

- Restrições a nível de Tabela

```
CREATE TABLE NotaVenda (
IdNV CHAR(6) NOT NULL,
DataNV DATE,
CONSTRAINT PK_NotaVenda PRIMARY KEY(IdNV));

CREATE TABLE Produto (
IdProd CHAR(5) NOT NULL,
DescProd VARCHAR(100),
PUnitProd DECIMAL(6,2),
QuantProd INTEGER,
TipoProd CHAR(12),
CONSTRAINT PK_Produto PRIMARY KEY(IdProd),
CHECK (TipoProd IN ('HARDWARE','SOFTWARE')));

CREATE TABLE ItemNota (
IdNV CHAR(6) NOT NULL,
IdProd CHAR(5) NOT NULL);
```

- Restrições a nível de coluna

```
CREATE TABLE NotaVenda (
IdNV CHAR(6) NOT NULL PRIMARY KEY,
DataNV DATE
);

CREATE TABLE Produto (
IdProd CHAR(5) NOT NULL PRIMARY KEY,
DescProd VARCHAR(100),
PUnitProd DECIMAL(6,2),
QuantProd INTEGER,
TipoProd CHAR(12) CHECK (TipoProd IN ('HARDWARE','SOFTWARE'))
);

CREATE TABLE ItemNota (
IdNV CHAR(6) NOT NULL PRIMARY KEY
REFERENCES NotaVenda ON DELETE CASCADE,
IdProd CHAR(5) NOT NULL
REFERENCES Produto
);
```

ALTERANDO DE TABELAS

Sintaxe:

```
ALTER TABLE nome_tabela <operação> [, <operação>];
<operação> = {ADD <definição_coluna>
| ADD <restrição_tabela>
| ALTER [COLUMN] nome_coluna
{TO novo_nome | TYPE novo_tipo | POSITION nova_posição}
| DROP CONSTRAINT nome_restrição}
```

Não há no ALTER TABLE a opção "DROP COLUMN". Para fazer alterações na definição de tabelas, deve-se:

- Armazenar o conteúdo da tabela em uma tabela temporária;
- Eliminar a tabela antiga;
- Definir a nova tabela;
- Carregar a nova tabela a partir da tabela temporária.

Exemplo(s):

- Adicionando ou modificando colunas de uma tabela

```
ALTER TABLE ItemNota add constraint PK_ItemNota PRIMARY KEY(IdNV,IdProd);

ALTER TABLE Produto
ADD CodBarraProd VarChar(20) NOT NULL;

ALTER TABLE NotaVenda
ADD NomeVendedor VarChar(50);
```

- Adicionando ou removendo uma restrição de uma tabela

```
ALTER TABLE Produto
ADD CONSTRAINT CodBarraProd_Produto_UK UNIQUE (CodBarraProd);

ALTER TABLE `aula1`.`itemnota`
ADD CONSTRAINT `FK_Itemnota_NotaVenda`
  FOREIGN KEY (IdNV)
  REFERENCES `aula1`.`notavenda` (IdNV)
  ON DELETE CASCADE;

ALTER TABLE `aula1`.`itemnota`
ADD CONSTRAINT `FK_Itemnota_Produto`
  FOREIGN KEY (IdProd)
  REFERENCES `aula1`.`produto` (IdProd)
  ON DELETE CASCADE;
```

Para excluir uma chave estrangeira é necessário que ela tenha recebido um nome quando da sua definição.

REMOVENDO TABELAS

Sintaxe:

```
DROP TABLE name [CASCADE | RESTRICT];
```

Exemplo(s):

```
DROP TABLE PRODUTO;
```

ÍNDICES

Os índices são usados para acelerar a recuperação de dados em resposta a certas condições de pesquisa e reforçar as restrições de unicidade em colunas. Sugestões para criação de índices:

- Colunas usadas frequentemente em condições de pesquisa (cláusula WHERE no comando SELECT);
- FOREIGN KEYS, pois estão geralmente envolvidas em JOINS;

Não há necessidade de indexar colunas:

- PRIMARY KEYS e UNIQUE KEYS (normalmente o sistema já cria internamente um índice);
- Raramente farão parte de condição de pesquisa;
- Possuem um domínio de dados muito pequeno;

Observações:

- Índices não podem ser alterados; devem ser removidos (com DROP) e recriados;
- A decisão de se usar ou não um índice em resposta a uma solicitação específica de dado não é tomada pelo usuário mas sim pelo sistema;

CRIANDO ÍNDICES

Sintaxe:

CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]] INDEX nome_índice ON nome_tabela (nome_coluna [, nome_coluna]);

Exemplo(s):

```
CREATE INDEX DescProd_idx
ON Produto(DescProd);
```

REMOVENDO ÍNDICES

Sintaxe:

DROP INDEX nome_índice;

Exemplo(s):

DROP INDEX DescProd_idx;

DML – *DATA MANIPULATION LANGUAGE*

INSERINDO REGISTROS

Sintaxe:

```
INSERT INTO <nome_tabela> [(nome_coluna [, nome_coluna])]
{VALUES (<valor> [, <valor> ])
| <subquery>};
```

Inserir novas linhas em uma tabela

- Sem informar os nome dos campos

Exemplo(s):

```
INSERT INTO Produto
VALUES ('00010','Mouse duas cores',5.20,38,'HARDWARE','12345');

INSERT INTO Notavenda VALUES ('0001','2002/10/14','ANA');
INSERT INTO Notavenda VALUES ('0002','2004/10/14','JOANA');
INSERT INTO Notavenda VALUES ('0003','2003/08/14','ANA');
INSERT INTO Notavenda VALUES ('0004','2006/08/14','JOANA');
INSERT INTO Notavenda VALUES ('0005','2002/11/14','SILVANA');
INSERT INTO Notavenda VALUES ('0006','2005/12/14','SILVANA');
```

- Informando os nomes dos campos

Exemplo(s):

```
INSERT INTO Produto (idprod, descprod, punitprod, quantprod, tipoprod, codbarraprod)
VALUES ('00015','Drive CDROM', 50.00, 10, 'HARDWARE', '1122');
```

- Copiar linhas de uma outra tabela

Exemplo(s):

```
CREATE TABLE NotaVendaHistorico (
IdNV CHAR(6) NOT NULL PRIMARY KEY,
DataNV DATE);
```

```
INSERT INTO `aula1`.`notavendahistorico` (`IdNV`, `DataNV`) VALUES
('00123', '10/10/2005');

INSERT INTO NotaVendahistorico
SELECT IdNV, DataNV
FROM NotaVenda;

INSERT INTO NotaVendahistorico
SELECT IdNV, DataNV
FROM NotaVenda where idnv <> '0005';
```

REMOVENDO REGISTROS

Sintaxe:

```
DELETE FROM nome_tabela
[WHERE <condição>];
```

- Remove todas as linhas da tabela se for omitida a cláusula WHERE.

Exemplo(s):

```
DELETE FROM Produto
WHERE IdProd = '00010';

DELETE FROM depto;
```

ATUALIZANDO REGISTROS

Sintaxe:

```
UPDATE nome_tabela
SET nome_coluna = <valor> [, nome_coluna = <valor>]
[WHERE <condição>]
```

- Atualizar linhas de uma tabela

Exemplo(s):

```
UPDATE NotaVenda
SET NomeVendedor = 'JAIR BATISTA'
WHERE IdNV = '0002';

UPDATE Produto
SET PunitProd = (PunitProd * 1.1) where CodBarraProd='1122';
```

Obs.: Todas as linhas de uma tabela são atualizadas se for omitida a cláusula WHERE.

CONSULTANDO REGISTROS

Sintaxe:

```
SELECT [DISTINCT] nome_coluna [,nome_coluna...] FROM
nome_tabela
```

- Seleção de colunas específicas

  Basta relacionar as colunas desejadas.

Exemplo(s):

```
SELECT IdNV, NomeVendedor
FROM NotaVenda;
```

- Seleção de todas as colunas

  Substituir os nome das colunas por *.

```
SELECT *
FROM Produto;
```

- Evitando duplicações de linhas

  Usar a palavra DISTINCT na cláusula SELECT.

```
SELECT DISTINCT NomeVendedor
FROM NotaVenda;
```

- Usando Pseudônimos

Uma consulta SQL normalmente usa o nome da coluna como cabeçalho; é possível definir um pseudônimo para a coluna que aparecera no cabeçalho

```
SELECT IdNV 'Numero da Nota'
FROM NotaVenda;
```

- Operador AS

O operador AS é usado para gerar um novo nome para a coluna. Geralmente é usado quando o *dataset* resultante possui alguma coluna derivada.

```
SELECT DescProd AS NomeProduto
FROM Produto
```

- Uso cláusula WHERE

Usada para filtrar um subconjunto de linhas de uma tabela, com base em uma condição. A condição deve resultar em um valor booleano (Verdadeiro ou Falso).

Sintaxe:

```
SELECT nome_coluna [,nome_coluna...]
FROM nome_tabela
[WHERE condição]
```

- Operadores

=	Igual a
< >	Diferente de
>	Maior que
<	Menor que
Between <valor> and <valor>	Entre dois valores
In (lista)	Qualquer valor da lista
Like <valor>	Corresponde a um padrão
IS NULL	É valor Nulo

Exemplo(s):

```
SELECT IdProd, DescProd
FROM Produto
        WHERE TipoProd = 'HARDWARE';

SELECT *
FROM Produto
WHERE PUnitProd > 10.00

SELECT IdProd, DescProd, TipoProd
FROM Produto
WHERE PunitProd BETWEEN 5.00 AND 10.00;

SELECT IdProd, DescProd, TipoProd
FROM Produto
WHERE IdProd IN ('00010','00011','00015');

SELECT *
FROM NotaVenda
WHERE NomeVendedor IS NULL;
```

Observações:

- LIKE faz reconhecimento de padrões:

"_"	equivale a qualquer caracter
"%"	equivale a qualquer seqüência de caracteres

Selecionar todos os produtos com a palavra 'azul' na descrição.

```
SELECT *
FROM PRODUTO
WHERE TipoProd LIKE '%ard%';
```

- Os operadores BETWEEN, IN, LIKE e IS NULL podem ser negados através do operador NOT: NOT BETWEEN, NOT IN , NOT LIKE, IS NOT NULL

```
SELECT IdProd, DescProd, TipoProd
FROM Produto
WHERE IdProd NOT IN ('00010','00011')
```

- **Conjunção de Condições**

Várias condições podem ser conectadas na cláusula WHERE usando o operador AND.

Exemplo(s):

```
SELECT IdProd, DescProd
FROM Produto
WHERE (TipoProd = 'HARDWARE') AND (PunitProd > 15.00);
```

- **Disjunção de Condições**

Uso do operador OR.

Exemplo(s):

```
SELECT IdProd, DescProd
FROM Produto
WHERE (TipoProd = 'SOFTWARE') OR (DescProd LIKE 'M%');
```

- **Operações no resultado da consulta**

São permitidas as operações de adição, subtração, multiplicação e divisão. Podem ser usadas as colunas das tabelas e constantes.

Exemplo(s):

```
SELECT Idprod, PunitProd, QuantProd, (PunitProd * QuantProd) as ValorTotal
FROM Produto;

SELECT Idprod, PunitProd, (PunitProd * 1.1) as PcoComAumento
        FROM Produto;
```

- **Uso da cláusula ORDER BY**

A cláusula ORDER BY é usada para ordenar o resultado de uma consulta.

Sintaxe:

```
SELECT nome_coluna [,nome_coluna...]
FROM nome_tabela
[WHERE condição]
[ORDER BY {nome_coluna, ...} [ASC|DESC]]
- ASC (Ascending): Ordem crescente
```

- DESC (Descending): Ordem decrescente

Exemplo(s):

```
SELECT IdProd, DescProd
FROM Produto
WHERE TipoProd = 'HARDWARE'
ORDER BY DescProd;

SELECT *
FROM NotaVenda
ORDER BY DataNV DESC;
```

- **Junção de Tabelas (Join)**

INNER JOIN (default)

As linhas de uma tabela podem ser combinadas às linhas de outra tabela através de valores iguais em colunas correspondentes.

Sintaxe 1:

```
SELECT nome_coluna [,nome_coluna...]
FROM <nome_tabela alias>, <nome_tabela alias>
WHERE <condição_join>
```

Sintaxe 2:

```
SELECT nome_coluna [,nome_coluna...]
FROM <nome_tabela> [INNER] JOIN <nome_tabela> ON <condição_join>
```

Exemplos Sintaxe 1:

- Sem o uso de Aliases

```
SELECT IdNV, DescProd
FROM ItemNota, Produto
WHERE ItemNota.IdProd = Produto.Idprod
```

- Com o uso de Aliases

```
SELECT i.IdNV, p.DescProd
FROM ItemNota i, Produto p
WHERE i.IdProd = p.Idprod
```

Exemplo Sintaxe 2:

```
SELECT IdNV, DescProd
FROM ItemNota INNER JOIN Produto
ON ItemNota.IdProd = Produto.Idprod
```

OUTROS TIPOS DE JOIN

A operação OUTER JOIN é usada quando é necessário fazer um JOIN de duas tabelas, mesmo que não existam os valores correspondentes em uma das duas tabelas. Pode ser LEFT JOIN, RIGTH JOIN ou FULL JOIN.

A operação Self JOIN (join reflexiva) é usada quando é necessário fazer um JOIN de uma tabela com ela mesma, comparando linhas de dados dentro de uma única tabela.

Exemplo de Self Join:

```
Listar todos os produtos que têm o mesmo preço.
SELECT p1.DescProd, p2.DescProd
FROM Produto p1, Produto p2
WHERE p1.PUnitProd=p2.PUnitProd
AND p1.idProd < p2.Idprod
```

A comparação dos códigos de produtos (Idprod) serve apenas para eliminar duas linhas do mesmo produto e evitar que o mesmo par de produtos (p1,p2) apareça também na ordem inversa (p2,p1).

PRODUTO CARTESIANO

Todas as linhas da primeira tabela são combinadas com todas as linhas da segunda tabela. Ocorre quando são usadas duas tabelas na cláusula FROM e a cláusula WHERE é omitida.

Exemplo(s):

Tabela NotaVenda com 14 registros
Tabela Produto com 4 registros

```
SELECT IdNv,Idprod
FROM NotaVenda, Produto; {gera um resultado com 56 registros}
```

FUNÇÕES DE MANIPULAÇÃO DE VALORES

Como regra geral, as funções para manipulação de valores são dependentes do SGBD utilizado.

- **Operadores aritméticos**

+	Soma
-	Subtração
*	Multiplicação
/	Divisão

- **Funções de conversão**

UPPER(string)	Converte todos os caracteres da string para maiúsculo

- **Funções de agregação**

São funções do padrão SQL que trabalham sobre um grupos de linhas, retornando um único valor com resultado.

AVG(coluna)	Média dos valores da coluna
COUNT(coluna)	Quantidade de linhas
MIN(coluna)	Menor valor da coluna
MAX(coluna)	Maior valor da coluna
SUM(coluna)	Soma dos valores da coluna

Sintaxe:

```
SELECT função_grupo(nome_coluna)
FROM nome_tabela
[WHERE condição]
[ORDER BY nome_coluna]
```

Exemplo(s):

Quantas notas já foram emitidas?

```
SELECT COUNT(IdNV)
FROM NotaVenda;
```

Qual o valor total em estoque?

```
SELECT Sum(PunitProd * QuantProd)
FROM Produto;
```

Observações

- Se o comando SELECT contiver funções de grupo, o resultado será somente uma linha. Assim, no comando SELECT não podem aparecer resultados individuais junto com expressões que contenham funções de grupo. Neste caso deveríamos usar GROUP BY. Por exemplo, o seguinte comando não é permitido:

```
SELECT IdNV, Count(IdProd)
FROM ItemNota;
```

- Todas as funções de agregação excluem linhas que tenham valor NULO na coluna que está sendo agregada.
- Por padrão estas funções incluem valores duplicados nos cálculos. Para excluir duplicidades, deve ser usada a palavra DISTINCT antes do nome da coluna.

Exemplo(s):

Quantos produtos diferentes já foram vendidos?

```
SELECT COUNT(IdProd)
FROM ItemNota; ---->> Errado, pois conta produtos duplicados
```

Correto:

```
SELECT COUNT(DISTINCT IdProd)
FROM ItemNota;
```

- Para contar todas as linhas de uma tabela pode ser usado COUNT(*) (linhas com valores NULOS também serão consideradas).

- Quando é usada a cláusula WHERE, primeiramente é feita a restrição das linhas (indicada pelo WHERE) e somente depois a função de agregação é aplicada.

Exemplo(s):

Quantas Notas foram emitida pelo vendedor JOSE?

```
SELECT COUNT(IdNV)
FROM NotaVenda
WHERE VendedorNome = 'JOSE';
```

Uso da Cláusula GROUP BY

Na grande maioria dos casos, as funções de agregação são usadas em cálculos baseados em grupos de linhas da tabela. A cláusula GROUP BY é usada para dividir as linhas de uma tabela em subgrupos menores.

O SQL recupera cada grupo de linhas de acordo com os valores da(s) expressão(ões) especificada(s) na cláusula GROUP BY.

Sintaxe:

```
SELECT nome_coluna [, nome_coluna ... ], função_de_grupo(nome_coluna)
FROM nome_tabela
[WHERE condição]
[GROUP BY expr1, expr2, ...]
```

A cláusula GROUP BY deverá vir sempre após a cláusula WHERE (ou após a cláusula FROM quando não existir WHERE).

Quando a cláusula GROUP BY é utilizada, é possível combinar resultados individuais com funções de grupo na cláusula SELECT, desde que aqueles resultados individuais sejam usados no GROUP BY.

Exemplo(s):

Quantas notas de venda foram emitidas por cada vendedor?

```
SELECT NomeVendedor, COUNT(IdNV)
FROM Notavenda
GROUP BY NomeVendedor;
```

Qual a quantidade de produtos por tipo?

```
SELECT TipoProd, COUNT(IdProd)
FROM Produto
GROUP BY TipoProd;
```

Qual o valor em estoque por tipo de produto?

```
SELECT TipoProd, Sum(PunitProd * QuantProd)
FROM Produto
GROUP BY TipoProd;
```

Qual o valor total de cada nota de venda? (Utilização de join)

```
SELECT i.IdNV, Sum(p.PunitProd * i.Quant)
FROM ItemNota i, Produto p
WHERE i.IdProd=p.IdProd
GROUP BY TipoProd;
```

Observações

- Quando usar GROUP BY todas as colunas que são usadas no SELECT mas não são usadas na função de grupo devem ser incluídas na clausula GROUP BY.
- Usando a cláusula WHERE pode-se selecionar as linhas antes de agrupá-las.

Exemplo(s):

Usando GROUP BY para múltiplas colunas:

Qual a quantidade de notas emitidas em cada dia por cada vendedor?

```
SELECT DataNV, VendedorNome, COUNT(IdNV)
FROM NotaVenda
GROUP BY DataNV, VendedorNome;
```

Uso da cláusula GROUP BY com HAVING

A cláusula WHERE não pode ser usada para restringir funções de grupo. Assim, o exemplo seguinte não é válido:

```
SELECT num_depto, AVG(salario_emp)
FROM emp
WHERE AVG(salario_emp) > 2000
GROUP BY num_depto
```

; ---->>> Errado: WHERE com função de grupo

Os grupos definidos pela cláusula GROUP BY podem ser filtrados pela cláusula HAVING.

Sintaxe:

```
SELECT nome_coluna [, nome_coluna ... ], função_de_grupo(nome_coluna)
FROM nome_tabela
[WHERE condição]
[GROUP BY expr1, expr2...]
[HAVING função_de_grupo(nome_coluna)]
[ORDER BY nome_coluna]
```

Exemplo(s):

Quais os vendedores já emitiram mais de uma nota?

```
SELECT NomeVendedor, COUNT(IdNV)
FROM Notavenda
GROUP BY NomeVendedor
HAVING COUNT(idNv) > 1;
```

Quais as notas de venda em que a quantidade de produtos vendidos é superior a 5?

```
SELECT IdNV, SUM(Quant)
FROM ItemNota
GROUP BY idNV
HAVING SUM(Quant) > 3;
```

O processamento da instrução SQL pode ser visto da seguinte forma:

- É feito o produto cartesiano das tabelas envolvidas;
- São selecionadas as linhas da tabela que obedecem ao critério da cláusula WHERE;
- São criados grupos de linhas que contenham valores idênticos nas colunas do GROUP BY;

- São selecionados os grupos que atendem ao critério da cláusula HAVING;
- É feita a classificação do resultado pelos valores das colunas da cláusula ORDER BY;
- É feita a projeção sobre as colunas especificadas no SELECT.

SUBQUERIES

Subqueries são comandos SELECT utilizados em condições de cláusulas WHERE ou HAVING, para prover resultados que são utilizados para completar a consulta principal.

Uma subquery que retorna apenas uma linha como resultado é chamada "single-row subquery". Caso mais de uma linha seja retornada, a subquery é chamada "multiple-row subquery".

Os operadores = , > , >= , < , <= e <> podem ser usados em comparações com "singlerow subqueries". Os operadores IN, ANY e ALL são usados em "multiple-row subqueries".

Exemplo(s):

Listar a descrição dos produtos que têm o mesmo preço unitário que o produto '00010'.

```
SELECT DescProd
FROM Produto
WHERE PunitProd = (SELECT PUnitProd
FROM Produto
WHERE IdProd = '00010');
```

Listar o nome dos vendedores que emitiram tantas notas de venda quando a vendedora 'PAULO'.

```
SELECT NomeVendedor
FROM NotaVenda
GROUP BY NomeVendedor
HAVING COUNT(IdNv) = (SELECT COUNT(IdNv)
FROM NotaVenda
WHERE NomeVendedor = 'PAULO');
```

- **Operadores ANY e ALL**

Quando a subquery retornar mais de um valor, os operadores ANY e ALL podem ser utilizados para compatibilizar o resultado da subquery com o tipo do operador de comparação.

<campo> > ANY <subquery>

Essa condição será verdadeira quando <campo> for maior que qualquer um dos resultados da subquery.

<campo> < ANY <subquery>

Essa condição será verdadeira quando <campo> for menor que qualquer um dos resultados da subquery.

<campo> > ALL <subquery>

Essa condição será verdadeira quando <campo> for maior que todos os resultados da subquery.

<campo> < ALL <subquery>

Essa condição será verdadeira quando <campo> for menor que todos os resultados da subquery.

Exemplo(s):

- Listar os produtos que tenham preço unitário superior ao preço unitário de todos os produtos de hardware.

```
SELECT Descprod
FROM Produto
WHERE Punitprod > ALL ( SELECT Punitprod
FROM Produto
WHERE Tipoprod = 'HARDWARE');
```

- **Operadores IN e NOT IN**

Verifica se o dado faz parte ou não da lista fornecida. A lista pode ser formada por valores retornados por uma subquery.

Exemplo(s):

Listar as notas de venda e os respectivos vendedores, em que constem mais de 2 produtos vendidos.

```
SELECT IdNV, NomeVendedor
FROM NotaVenda
WHERE IdNV IN (SELECT IdNV
FROM ItemNota
GROUP BY IdNV
HAVING COUNT(IdProd) > 2)
```

- **Operadores de Conjuntos**

Como o resultado de um query é um conjunto de linhas você pode realizar operações de conjuntos entre queries.

- UNION : União entre os resultados das queries; •
  INTERSECT : Interseção entre os resultados das queries;
- MINUS : Subtração entre os resultados das queries.

A operação de união permite reunir os resultados de duas consultas distintas em um só resultado.

Equivale à operação de União da Álgebra Relacional. A operação de união elimina as linhas duplicadas.

Exemplo(s):

Listar todas as Notas de venda que sejam do vendedor 'JUCA', ou em que conste o produto '00014'.

```
SELECT IdNV FROM NotaVenda WHERE NomeVendedor='JUCA'
UNION
SELECT IdNV FROM ItemNota WHERE IdProd='00014'
```

- **Operadores EXISTS e NOT EXISTS**

EXIST: retorna “verdadeiro” se uma determinada subquery retornar ao menos uma linha e “falso” caso contrário.

NOT EXIST: retorna o resultado contrário do operador EXIST.

Exemplo(s):

Listar a descrição de todos os produtos que já foram vendidos.

```
SELECT DescProd
FROM Produto p
WHERE EXISTS ( SELECT IdProd FROM ItemNota i WHERE p.IdProd = i.IdProd
)
```