CENTRO DO TREINAMENTO PROFISSIONAL UBUNTU SOCIEDADE

BASE DE DADOS

FORMADOR: Engº Bernardo Kianvuidi Paulo Lukoki
CONTACTO
Phone: 938389866
Email: paulo-g11@hotmail.com

07/12/2022

I.I. DADOS

É materia prima que uma vez transformada se obtem uma informação, é tida como a menor particula que constitui uma informação.

I.2. PROCESSAMENTO

É o processo de tratamento ou tranformação automática dos dados.

I.3. INFORMAÇÃO

É o resultado obtido após o processamento dos dados.

2. BASES DE DADOS

Diz-se um conjunto de dados interligados internamente entre uns aos outros por meio de entidades.

Agrupado de dados em diferentes entidades que interagem entre si.

A maneira com que os dados (intidades) se interligam, definem os modelos de bases de dados.

2.I. MODELOS DE BASES DE DADOS

Os modelos existem vários, nomeadamente:

- Em Redes;

- Hierarquicos

- Orientado a Objeto

- Relacional

- Entidade – Relacionamento

Obs: O modelos amplamente abordado, é a da entidade e relacionamento.

2.I.I. MER – MODELO ENTIDADE – RELACIONAMENTO

É um modelo utilizado para descrever os objetos do mundo real através de **entidades**, com suas propriedades que são os **atributos** e os seus **relacionamentos**.

De forma simples, podemos dizer que o Modelo de Entidade e Relacionamento (**MER**) são diagramas utilizados para projetar Bancos de Dados Relacionais, utilizando como base a relação de objetos reais, e sendo representado por meio de entidades e relacionamentos.

2.I.I.I. ENTIDADE

Entidades são abstrações de objetos do mundo real, representados que são representadas nominalmente por substantivos. Alguns exemplos de entidades são: Pessoas, Automóveis, Departamentos, entre outros.

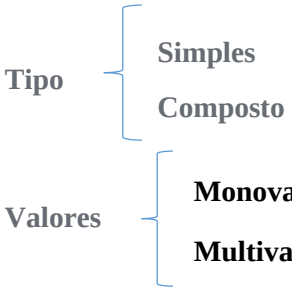
2.1.1.2. ATRIBUTOS

Atributos são características de uma entidade. Exemplos de atributos para as entidades abaixo são:

Entidade	Exemplos de Atributos
Pessoas	CPF, Nome, Data de Nascimento, Telefone e Endereço
Automóveis	Nome do Automóvel, Chassi, Placa e Cor

2.1.1.2.1. CLASSIFICAÇÃO DOS ATRIBUTOS

Os atributos são classificados em tipos e valores



- **Atributos Simples:** Possuem apenas uma propriedade associada à entidade, por exemplo, o atributo nome da entidade pessoa;
- **Atributos Compostos:** Possuem mais propriedades associadas ao mesmo atributo, por exemplo, é o caso do endereço associado à entidade pessoa, pois ele possui os atributos logradouro, endereço, número, complemento, bairro e CEP;
- **Atributos Monovalorados:** Possui um único valor para a entidade, como por exemplo, o campo nome relacionado à entidade pessoa;
- **Atributos Multivalorados:** São atributos que possuem mais de um valor. Por exemplo, o caso do telefone associado à entidade pessoa, pois é possível não ter nenhum telefone ou ter vários.

Ainda, é possível classificar os atributos como:

- Determinante;
- Derivado;
- Tipo de atributo;
- Domínio do atributo.

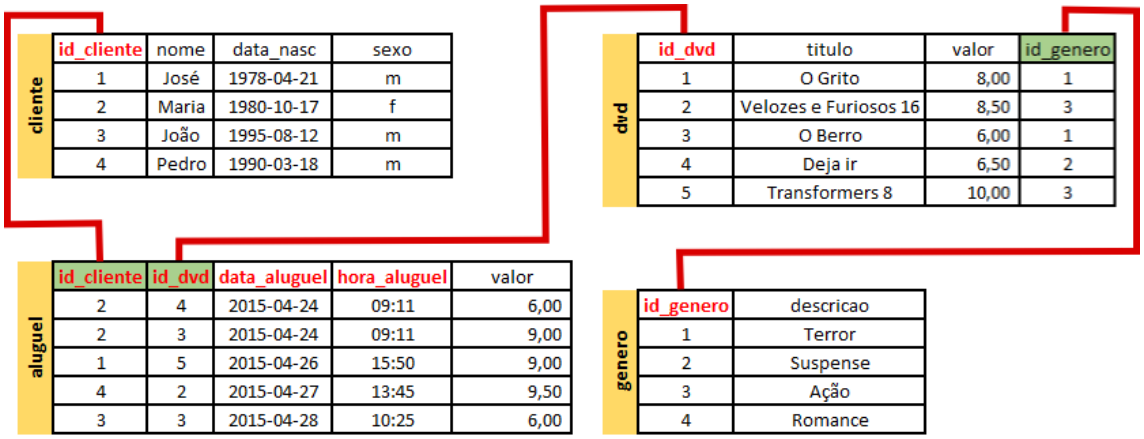
- Determinante: Atributo para identificar um dado relacionado à entidade como sendo único, como o caso do NºBI relacionado à entidade pessoa, por exemplo;

  Os **atributos determinantes** serão as chaves primárias no **banco de dados** e seu uso tem implicações na normalização de **dados**.

  Atributo chave é utilizado para identificar de forma única uma entidade, ou seja, os valores associados a esse **atributo são** distintos dentre o conjunto de entidades.

  Chave primária, ou Primary key (PK) : é o identificador único de um registro na tabela. Pode ser constituída de um campo (**chave** simples) ou pela combinação de dois ou mais campos (**chave** composta), de tal maneira que não existam dois registros com o mesmo valor de **chave primária**.

- **Não pode ter duplicação –** o campo que for definido como chave primária não pode ser igual em dois registos diferentes;
- **Não pode ser nula –** o campo que é chave primária tem que ter um valor inserido. Não pode ser nulo, não assumir valor zero;
- **A primeira tabela –** conjunto de dados – dum ficheiro tem de ter pelo menos uma chave primária atribuída a um campo para identificar inequivocamente os registos da tabela. A segunda e seguintes tabelas dum ficheiro podem ter, ou não, chaves primárias atribuídas.



cliente

id_cliente	nome	data_nasc	sexo
1	José	1978-04-21	m
2	Maria	1980-10-17	f
3	João	1995-08-12	m
4	Pedro	1990-03-18	m

dvd

id_dvd	titulo	valor	id_genero
1	O Grito	8,00	1
2	Velozes e Furiosos 16	8,50	3
3	O Berro	6,00	1
4	Deja ir	6,50	2
5	Transformers 8	10,00	3

aluguel

id_cliente	id_dvd	data_aluguel	hora_aluguel	valor
2	4	2015-04-24	09:11	6,00
2	3	2015-04-24	09:11	9,00
1	5	2015-04-26	15:50	9,00
4	2	2015-04-27	13:45	9,50
3	3	2015-04-28	10:25	6,00

genero

id_genero	descricao
1	Terror
2	Suspense
3	Ação
4	Romance

Chave Candidata: é uma chave que parece ser a primária mais não é.

Uma **chave candidata** é um identificador único dentro de uma relação que garante que nenhuma tupla será duplicada. Vale ressaltar que uma chave candidata não necessariamente é uma chave primária: uma tupla pode ter várias chaves candidatas mas só uma chave primária. Nesse quesito, uma tupla de usuário onde existem os campos "ID" e "BI" tem uma chave primária (ID) e três chaves candidatas. Uma chave candidata pode ser formada por vários atributos.

Chaves candidatas ocorrem quando em uma relação existe mais de uma combinação de atributos para a identificação única do registro.

Exemplo: ALUNO (id, numbi ...)

Chave estrangeira

A chave estrangeira ocorre quando um atributo de uma relação for chave primária em outra relação. Em outras palavras sempre que houver o relacionamento 1:N entre duas tabelas, a tabela 1 receberá a chave primária e a tabela N receberá a chave estrangeira.

A importância do uso de umna chave primária

A **chave primária** é fundamental e tem por função não só **ser** o identificador único do registro na tabela mas também permitir a ligação com outras tabelas, onde esta **se** tornaria a **chave estrangeira**.

Integridade referencial é um conceito relacionado à **chaves** estrangeiras. Este conceito diz que o valor que é **chave** estrangeira em uma tabela destino, deve ser **chave primária**de algum registro na tabela origem. Quando essa regra é desrespeitada, então temos o caso em que a **integridade referencial** é violada.

Vejamos a terminologia: Integridade vem de íntegro, inteiro, completo, correto. Referencial vem de referência, indicar algo ou alguém. Portanto, integridade referencial é indicar algo ou alguém de forma íntegra, completa, correta

Carros

Placa (PK)	Modelo	Proprietário (FK)
ABC-1233	I10	1
DEF-4566	Rav4	2
UUV-7890	Starlet	1

A tabela tem integridade referencial, pois os carros que têm donos com ID 1, podem ser encontrados na tabela de proprietários como sendo do Bernardo. O carro de proprietário com ID 2 pode ser encontrado como sendo da Celestina.

Proprietários

ID (PK)	Nome
1	Bernardo
2	Celestina

Agora, imagine que nós venhamos inserir um carro de placa EJB-6520, do modelo Celta e do proprietário com o ID 3. Ocorre que não há nenhum proprietário de ID 3. Se o banco de dados permitir essa inclusão, ocorrerá uma violação da integridade referencial, pois estará sendo feita uma referência a uma entidade inexistente. O mesmo ocorreria se quisermos alterar o proprietário de um dos carros colocando o ID do proprietário como 3.

Por outro lado, se nós quisermos deletar a Maria do banco de dados sem deletar o carro de placa DEF-4566 e nem alterá-lo, novamente teremos uma violação da integridade referencial, pois se o banco de dados permitir que essa exclusão seja feita, a integridade referencial será violada ao termos um carro que tem como dono, uma entidade agora inexistente.

- Atributo Derivado: O atributo serve como base para gerar outros atributos, por exemplo, o caso da idade relacionado à entidade pessoa que é calculado por meio do atributo data de nascimento;

- Tipo de Atributo: Cada atributo possui um tipo de dado relacionado, como por exemplo, textos, números inteiros, números decimais, datas, entre outras;

- Domínio do Atributo: Permite restringir o conjunto de dados permitidos ao atributo, como para o UF do endereço, utilizando somente os dados SP, PR, SC, RS, MG, por exemplo.

2.1.1.3. RELACIONAMENTO

São as relações criadas entre as entidades e elas são representadas por verbos.

2.1.1.4. CARDINALIDADE

é um número que expressa o comportamento (número de ocorrências) de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento

Nas cardinalidades temos:

- **Cardinalidade** Máxima: Número máximo de vezes que uma entidade **A** pode ocorrer em **B**. Pode assumir o valor de 1 ou N (inúmeras vezes).
- **Cardinalidade** Mínima: Número mínimo de vezes que uma entidade **A** pode ocorrer em **B**. Pode assumir o valor de 0 ou 1.

De acordo com a cardinalidade existem 3 tipos básicos de relacionamentos entre as entidades, nomeadamente:

- Relacionamento de cardinalidade 1:1: Denominado "um para um", é usado quando um elemento da entidade X se relaciona com um elemento da entidade Y;
- Relacionamento de cardinalidade 1:n: Denominado "um para muitos", é usado quando um elemento da entidade X se relaciona com um ou mais elementos da entidade Y;
- Relacionamento de cardinalidade m:n: Denominado "muitos para muitos", é quando vários elementos da entidade X se relacionam com um ou mais elementos da entidade Y.

OBS: Cardinalidade está relacionado com número de vezes que um entidade entra em outra entidade.

3. NORMALIZAÇÃO

Normalização **é um** processo a partir do qual se aplicam regras a todas as tabelas do banco de **dados** com o objetivo de evitar falhas no projeto, como redundância de **dados e** mistura de diferentes assuntos numa mesma tabela.

A normalização de dados é uma série de passos que se segue no projeto de uma base de dados que permite um armazenamento consistente e um eficiente acesso aos dados numa base de dados relacional. Esses dados reduzem a redundância de dados e as possibilidades dos dados se tornarem inconsistentes.

Inicialmente foram estabelecidas três formas normais:

- 1.ª Forma Normal (1FN);
- 2.ª Formal Normal (2FN);
- 3.ª Forma Normal (3FN).

3.1. Primeira Forma Normal

A normalização de uma tabela na 1.ª Forma Normal (1FN) exige que a tabela tenha uma estrutura bidimensional correta, ou seja, cada linha deve corresponder a um só registo e cada coluna a um só campo.

Além disso, cada campo deve conter dados atómicos, ou seja, um só dado por cada registo. Cada dado só deve tomar um valor dentro do domínio definido para esse campo.

CodCliente	Cliente	Morada	Encomendas			
			N_Enc	Data	Produto	Quantidade
C01	Aníbal	Lisboa	1	2010-05-25	Ananás	10
			5	2010-05-30	Cebolas	20
C02	Belmiro	Braga	3	2010-05-26	Bananas	30
C03	Casimiro	Coimbra	2	2010-05-25	Tomates	50
			4	2010-05-26	Cebolas	20

Consideremos a tabela acima como exemplo. Nesta tabelas, estão registadas encomendas feitas por clientes; mas a tabela não tem uma estrutura bidimensional correta (segundo a 1FN). Por exemplo, na linha onde está o registo do cliente Aníbal, temos dois conjuntos de encomendas; deveria ser um registo por linha.

Por outro lado, a coluna com o nome Encomendas abrange quatro campos; deveria existir somente um campo por coluna.

Corrigindo então, a tabela correta de modo a apresentar-se na 1FN segue-se abaixo.

CodCliente	Cliente	Morada	N_Enc	Data	Produto	Quantidade
C01	Aníbal	Lisboa	1	2010-05-25	Ananás	10
C01	Aníbal	Lisboa	5	2010-05-30	Cebolas	20
C02	Belmiro	Braga	3	2010-05-26	Bananas	30
C03	Casimiro	Coimbra	2	2010-05-25	Tomates	50
C03	Casimiro	Coimbra	4	2010-05-26	Cebolas	20

FONTE: https://filipacardosoblog.wordpress.com/normalizacao-de-uma-base-de-dados-tres-formas-normais/

Contudo, esta tabela, estando na 1FN, apresenta problemas de redundância de informação, ou seja, repetição desnecessária de alguns dados.

3.1.1. Problemas das Tabelas na IFN

As tabelas normalizadas apenas na 1FN podem apresentar problemas de redundância de informação, bem como outros tipos de problemas ou anomalias, nomeadamente:

- **anomalias de inserção** – em certas situações, a inserção de um novo registo pode implicar um ou mais campos em branco; por exemplo, se quisermos registar um novo cliente sem que, nesse momento, ele tenha feito qualquer encomenda, vários campos ficariam vazios.
- **anomalias de atualização** – dado que numa tabela pode existir repetição de informação sobre um mesmo elemento, se quisermos atualizar um dado relativo a um determinado elemento (por exemplo a morada de um cliente), teremos de efetuar tantas atualizações quantas as vezes que esse elemento aparecer na tabela.
- **anomalias de eliminação** – pela mesma razão anteriormente apontada, ou seja, a possibilidade de existir informação repetida sobre um mesmo elemento, se quisermos apagar um determinado elemento da nossa base de dados, teremos de o fazer a todos os registos em que ele figurar.

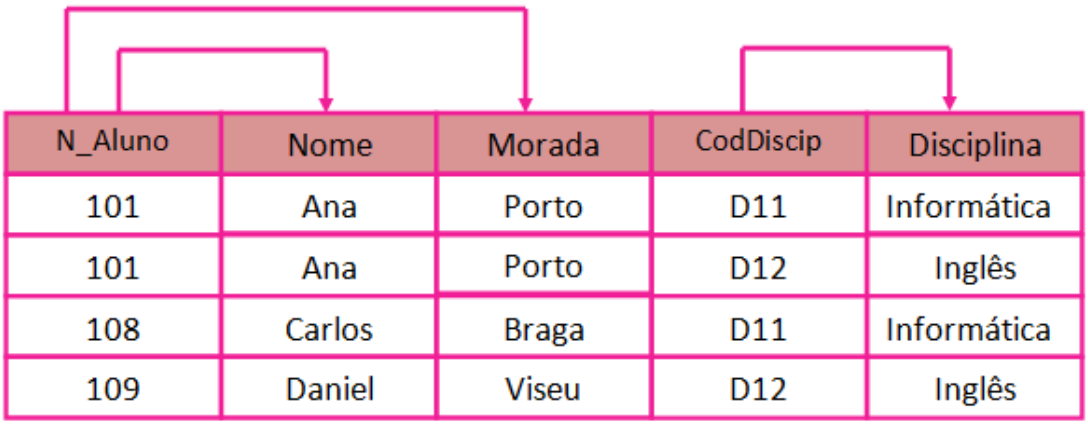
Os **problemas de atualização e eliminação** significam uma de duas coisas ou ambas:

- trabalho repetitivo que pode ser evitado;
- possibilidade de erros em que a base de dados perde consistência.

3.1.2. Dependências Funcionais

A compreensão da 2.ª e da 3.ª formas normais implica a percepção do conceito de **dependência funcional**.

Consideremos a seguinte tabela abaixo com os dados dos alunos e das disciplinas em que eles se encontram inscritos: Alunos (N_Aluno, Nome, Morada, CodDiscip, Disciplina).



N_Aluno	Nome	Morada	CodDiscip	Disciplina
101	Ana	Porto	D11	Informática
101	Ana	Porto	D12	Inglês
108	Carlos	Braga	D11	Informática
109	Daniel	Viseu	D12	Inglês

Como cada aluno pode inscrever-se em várias disciplinas, cada inscrição dá origem a um registo diferente na nossa tabela. Neste caso, só podemos diferenciar um registo de outro através do par atributos "N_Aluno, CodDiscip".

Assim, a **chave primária** da tabela Alunos é composta por dois campos: N_Aluno + CodDiscip, visto que só esse par identifica de modo unívoco os registos da tabela.

Quando um atributo ou conjunto de atributos identifica de modo unívoco os registos de uma tabela, diz-se que **determina funcionalmente** os outros atributos. Isso acontece sempre com a chave primária de uma tabela. Mas também pode acontecer em relação a outros campos.

Na tabela Alunos, o conjunto de atributos N_Aluno + CodDiscip **determina funcionalmente** os outros atributos (Nome, Morada, Disciplina).

Reciprocamente, diz-se que os atributos Nome, Morada, Disciplina são **funcionalmente dependentes** do par de atributos N_Aluno + CodDiscip.

Mas, além disso, podemos constatar que existem outras **dependências funcionais**:

1. Os atributos Nome e Morada são funcionalmente dependentes de N_Aluno (sabendo N_aluno, sabem-se Nome e Morada);
2. O atributo disciplina é funcionalmente dependente de CodDiscip (sabendo CodDiscip, sabe-se também o nome da disciplina).

3.2. Segunda Forma Normal

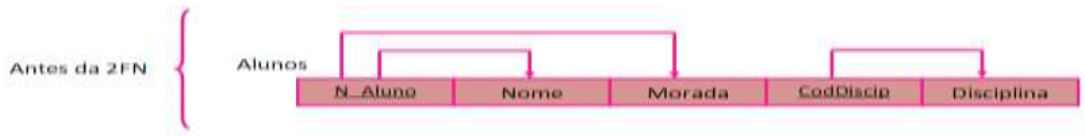
A **2.ª Forma Normal** diz que a tabela tem de estar na 1FN e que cada atributo não chave tem de ser funcionalmente dependente da totalidade da chave primária e não apenas de uma parte dessa chave.

Assim depois de identificarmos a chave primária de uma tabela, pode dar-se um dos dois casos:

1. **A chave primária é constituída por um só atributo** (chave elementar) – neste caso, a **tabela está seguramente na 2FN** (nenhum atributo depende de uma parte da chave, visto que a chave não e composta por partes);
2. **A chave primária é constituída por mais que um atributo** (chave primária composta) – neste caso, se existe algum ou alguns atributos que dependem de uma parte da chave (ou seja, de algum atributo que constitui a chave), então a **tabela não está na 2FN**.

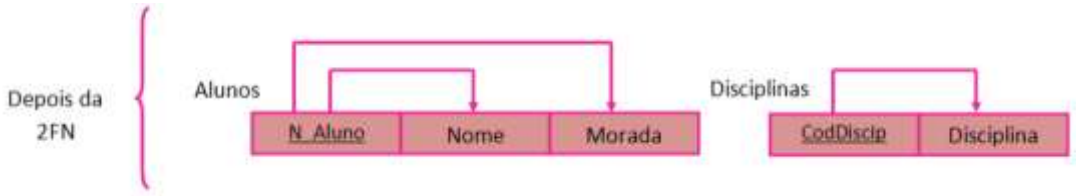
Considerando de novo a tabela Alunos, recordemos que a **chave primária** desta tabela é composta por N_Aluno CodDiscip. Verifica-se que temos atributos não chave que dependem funcionalmente de parte da chave:

- Nome e Morada dependem de N_Aluno;
- Disciplina depende de CodDiscip.



Para normalizar a referida tabela de acordo com a 2FN, teremos de a dividir em duas tabelas:

- Alunos (N_Aluno, Nome, Morada);
- Disciplinas (CodDiscip, Disciplina).



Dessa forma, cada uma das tabelas resultantes passa a ter uma **chave primária elementar**, constituída por um só atributo o que é suficiente para que essas tabelas se encontrem na 2FN: todo o atributo não chave depende da totalidade da chave (a chave não é composta por partes).

Entretanto, para podermos relacionar as tabelas Alunos e Disciplinas, temos que criar uma terceira tabela; por exemplo, assim:

- Aluno_Disciplina (N_Aluno, CodDiscip, DataInsc).

Os **relacionamentos entre alunos e disciplinas** serão efetuados através dos seguintes dois campos (com dados comuns):

- N_Aluno – que é a chave primária na tabela Alunos e chave externa em Aluno_Disciplina;
- CodDiscip – que é a chave primária na tabela Disciplinas e chave externa em Aluno_Disciplina.

Alunos

N_Aluno	Nome	Morada
101	Abel	Aveiro
101	Ana	Porto
103	Carlos	Braga
104	Daniel	Viseu

Disciplinas

CodDiscip	Disciplina
D11	Português
D12	Inglês
D13	Informática
D14	Matemática

N_Aluno	CodDiscip	DataInsc
101	D11	04-07-2010
101	D12	04-07-2010
102	D12	05-07-2010
102	D13	05-07-2010

Aluno_Disciplina

3.3. Terceira Forma Normal

A **3.ª Forma Normal** (3FN) diz que a tabela tem de estar na 2FN e que nenhum atributo não chave pode depender funcionalmente de algum outro atributo que não seja a chave primária.

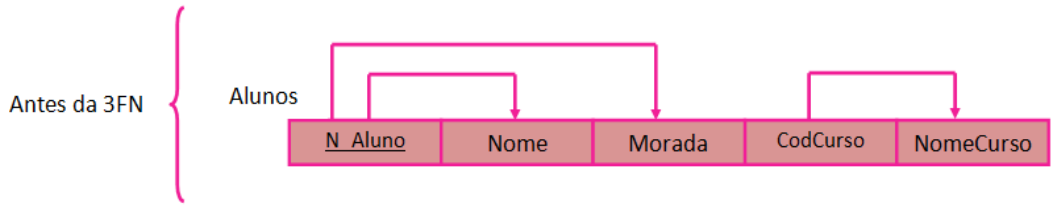
Portanto, para normalizar uma tabela de acordo com a 3FN, devemos analisar todos os atributos não chave e verificar se existem algumas dependências funcionais entre eles:

- se não existir nenhuma dependência funcional entre os atributos não chave, a tabela está na 3Fn;
- se existir alguma dependência funcional entre atributos não chave, então, é necessário retirar esse conjunto de atributos da tabela e construir com eles uma tabela à parte.

Analisemos a tabela Alunos:

-Alunos (N_Aluno, Nome, Morada, CodCurso, NomeCurso)

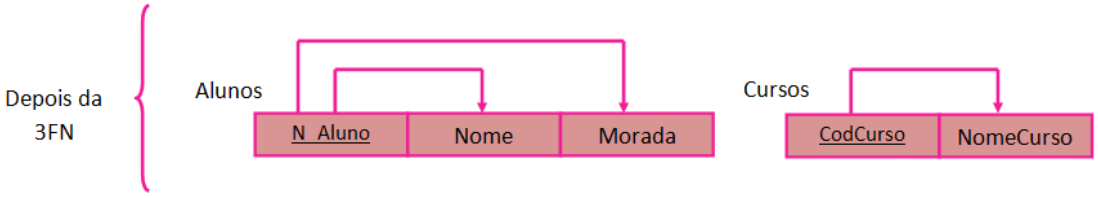
Nesta tabela, a chave primária é só N_Aluno. Cada aluno só pode estar inscrito num curso.



Podemos constatar que existe um atributo não chave que é funcionalmente dependente de um outro atributo: NomeCurso depende do campo CodCurso. Dado um código de um curso, sabemos qual o nome do curso, pois este depende funcionalmente daquele.

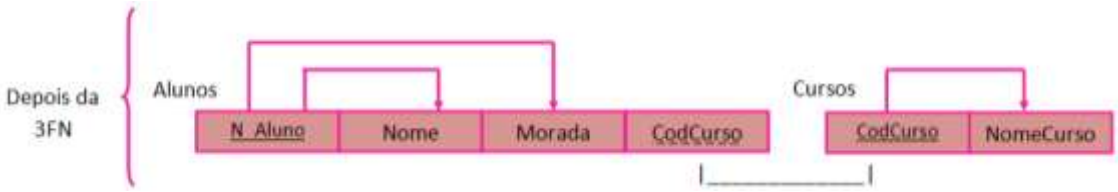
Nestas situações de dependências funcionais, devemos retirar o ou os atributos que dependem funcionalmente de outro ou outros e constituir uma tabela em separado.

No caso em análise, o conjunto formado por CodCurso e NomeCurso deve constituir uma tabela independente.



Entretanto, na tabela Alunos deve continuar o campo CodCurso, pois é ele que vai permitir relacionar esta tabela com a nova tabela Cursos. Na verdade, o campo CodCurso é a chave primária da tabela Cursos e, na tabela Alunos, ele vai assumir o papel de chave estrangeira/externa. Quando

se regista, na tabela Alunos, o código do curso que um aluno frequenta, esse código remete para a tabela Cursos, onde estão os nomes dos respetivos cursos.



4. IMPLEMENTAÇÃO DE BASES DE DADOS

O **banco de dados** é a organização e armazenagem de informações sobre um domínio específico. De forma mais simples, é o agrupamento de **dados** que tratam do mesmo assunto, e que precisam ser armazenados para segurança ou conferência futura.

4.I. O processo de estruturação

O processo de estruturação consiste nos seguintes passos:

1° - Determinar o objetivo da base de dados

Isto ajuda-o a preparar-se para os restantes passos.

2° - Localizar e organizar as informações necessárias

Reúna todos os tipos de informações que pretende registar na base de dados, como nomes de produtos e números de encomenda.

3° - Dividir informações em tabelas

Divida os itens de informação em entidades ou assuntos principais, como produtos ou encomendas. Cada assunto torna-se uma tabela.

4° - Transformar itens de informação em colunas

Decida que informações pretende armazenar em cada tabela. Cada item torna-se um campo e é apresentado como uma coluna na tabela. Por exemplo, uma tabela designada Funcionários poderá incluir campos como Apelido, Nome Próprio e Data de Contratação.

5° - Especificar chaves primárias

Escolha a chave primária de cada tabela. A chave primária é uma coluna utilizada para identificar cada linha. Por exemplo, pode ser o ID do Produto ou ID da Encomenda.

6° - Estabelecer relações de tabelas

Observe para cada tabela e decida de que forma é que os dados numa tabela estão relacionados com os dados noutras tabelas. Adicione campos às tabelas ou crie novas tabelas para definir as relações, conforme necessário.

7° - Aperfeiçoar a estrutura

Verifique se existem erros na estrutura. Crie as tabelas e adicione alguns registos de dados de exemplo. Veja se consegue obter os resultados que pretende a partir das tabelas. Ajuste a estrutura conforme necessário.

8° - Aplicar as regras de normalização

Aplique regras de normalização de dados para ver se as suas tabelas estão corretamente estruturadas. Ajuste as tabelas conforme necessário.

4.2. NIVEIS DE ABSTRAÇÃO DE DADOS

Visão dos dados (abstração dos dados)

- **Nível de visão ou Conceptual** – A abstração mais alta, descreve apenas ***parte*** do banco de dados.
- **Nível lógico** – O próximo nível de abstração, descreve ***quais*** dados estão armazenados no banco de dados e ***quais*** relações existem entre eles.
- **Nível físico** – Nível de abstração mais baixo, descreve ***como*** os dados são armazenados.

5. SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS (SGBD)

Um **Sistema de Gerenciamento de Banco de Dados** (português brasileiro) ou **Sistema de Gestão de Bases de Dados** (português europeu)(SGBD) — do inglês *Data Base Management System* (DBMS) — é o sistema de *software* responsável pelo gerenciamento de um ou mais bancos de dados. Seu principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, a persistência, a manipulação e a organização dos dados. O SGBD disponibiliza uma interface para que seus clientes possam incluir, alterar ou consultar dados previamente armazenados. Em bancos de dados relacionais a

interface é constituída pelas API (*Application Programming Interface*) ou *drivers* do SGBD, que executam comandos na linguagem SQL(*Structured Query Language*).

Um SGDB implica a criação e manutenção das bases de dados, elimina a necessidade de especificação de definição de dados, age como interface entre os programas de aplicação e os ficheiros de dados físicos, e separa as visões lógica e de concepção dos dados. Assim sendo, são basicamente três os componentes de um SGBD:

- Linguagem de definição de dados (especifica conteúdos, estrutura a base de dados e define os elementos de dados);
- Linguagem de manipulação de dados (para poder alterar os dados na base);
- Dicionário de dados (guarda definições de elementos de dados e respetivas caraterísticas — descreve os dados, quem os acede, etc.) (Gouveia; 2009).
    - Linguagem de definição de dados ou LDD (ou DDL, do inglês), com comandos como CREATE, DROP e ALTER TABLE;
    - Linguagem de manipulação de dados, ou LMD (ou DML, do inglês), com comandos como UPDATE, SELECT, INSERT e DELETE;
    - Linguagem de controle de dados, ou LCD, com comandos para controle de acesso dos usuários do sistema, como GRANT e REVOKE, em SQL.

▪ PostgreSQL ▪ CouchDB ▪ Firebird ▪ HSQLDB ▪ IBM DB2 ▪ IBM Informix ▪ mSQL ▪ MySQL ▪ MariaDB ▪ Oracle	▪ SQL-Server ▪ TinySQL ▪ ZODB ▪ JADE ▪ Sybase ▪ Microsoft Access ▪ Microsoft Visual Foxpro ▪ MongoDB

5.1. MICROSOFT ACCESS

Microsoft Access, conhecido por MSAccess, é um sistema de gerenciamento de banco de dados da Microsoft, incluído no pacote do Microsoft Office Professional, que combina o Microsoft Jet Database Engine com uma interface gráfica do utilizador. Wikipédia

1. Data de lançamento: novembro de 1992
2. Licença: Shareware
3. Desenvolvedor: Microsoft
4. Lançamento: novembro de 1992 (29–30 anos)

O **Access** permite a criação de formulários e cadastros completos com navegação, filtragem, ordenação e totalização dos dados através de grids, sub-formulários e pop-ups. O uso desta tecnologia não requer o envolvimento da área de TI, você pode contrata-lo como uma planilha automatica.

5.1.1. Usos como SGBD

Geralmente uma aplicação desenvolvida com o Access através da linguagem de programação VBA (Visual Basic for Applications) consiste em dois arquivos: Um que se denomina BackEnd, onde ficam armazenadas todas as tabelas com seus respectivos

relacionamentos, e outro denominado FrontEnd, onde ficam armazenados os códigos fontes, formulários, módulos, consultas, macros, etc.

O código fonte pode ser compilado, mas não é possível gerar um executável. Para rodar os aplicativos desenvolvidos é necessário que o usuário possua em sua estação de trabalho o **MSAccess** instalado ou pelo menos o seu **Access Runtime** que vem a ser uma versão semi completa do MSAccess que servirá apenas para rodar os aplicativos sem a possibilidade de desenvolvimento e poderá usar aplicativos criado em Access.

Com o **Microsoft Access** é possível desenvolver desde aplicações simples como por exemplo, um cadastro de clientes, controle de pedidos, até as aplicações mais complexas, como por exemplo, todo o controle operacional, administrativo e financeiro de uma pequena ou até mesmo de uma média ou grande empresa, pois os aplicativos desenvolvidos podem rodar perfeitamente numa rede de computadores e os dados armazenados pelo sistema podem ser publicados na Intranet ou até mesmo na Internet.

5.2. MICROSOFT SQL SERVER

É um gerenciador de bancos de dados relacional, o que significa dizer que as informações que manipula estão armazenadas em campos de tabelas. Para facilitar o entendimento, imagine, por exemplo, uma tabela de clientes, em que cada linha contém os registros de nome, endereço e estado civil.

O **Microsoft SQL Server** é um sistema gerenciador de Banco de dados relacional (SGBD) desenvolvido pela Sybase em parceria com a Microsoft.

Esta parceria durou até 1994, com o lançamento da versão para Windows NT e desde então a Microsoft mantém a manutenção do produto.

Como um Banco de dados, é um produto de software cuja principal função é a de armazenar e recuperar dados solicitados por outras aplicações de software, seja aqueles no mesmo computador ou aqueles em execução em outro computador através de uma rede (incluindo a Internet).

Há várias diferentes edições do Microsoft SQL Server destinadas a públicos diferentes e para diferentes cargas de trabalho (variando de pequenas aplicações que armazenam e recuperam dados no mesmo computador, a milhões de usuários e computadores que acessam grandes quantidades de dados a partir da Internet ao mesmo tempo).

Suas linguagens de consulta primárias são **Transact-SQL** (T-SQL) e ANSI SQL.

Mantido pela Microsoft há anos, é um dos principais SGBDs relacionais do mercado. Distribuído em diferentes edições e com várias ferramentas integradas, esse banco é capaz de atender às demandas desde os mais simples negócios até os mais complexos cenários que lidam com grande volume de dados.

Antigamente o SQL Server era um banco de dados relacional. Essencialmente você armazenava nele dados estruturados no formato tabular (linhas e colunas).

Ele permite a criação de tabelas relacionadas, evitando a necessidade de armazenar dados redundantes em vários locais dentro de um banco de dados. O modelo relacional também fornece integridade referencial e outras restrições de integridade para manter a precisão dos dados. O SQL Server suporta transações, é aderente e suporta os princípios de atomicidade, consistência, isolamento e durabilidade.

6. LINGUAGEM SQL

Structured Query Language, ou **Linguagem de Consulta Estruturada** ou **SQL**, é a linguagem de pesquisa declarativa padrão para banco de dados relacional (base de dados relacional). Muitas das características originais do SQL foram inspiradas na álgebra relacional.

A pesquisa `SELECT * FROM T`, no exemplo da tabela à direita acima, terá como resultado todos os elementos de todas as linhas da tabela chamada T. Partindo da mesma tabela T, a pesquisa `SELECT C1 FROM T` terá como resultado todos os elementos da coluna C1 da tabela T. O resultado da pesquisa `SELECT * FROM T WHERE C1=1` será todos os elementos de todas as linhas onde o valor de coluna C1 é '1'.

Exemplo 2

Exemplo básico de verificação se tabela existe em SQL:

```
SELECT *
```

```
        FROM
INFORMACAO.TABELA
        WHERE
ESQUEMA_TABELA = 'Esquema da Tabela'
        AND NOME_TABELA = 'Nome da Tabela')
```

Tabela 'T'	Consulta	Resultado
C1 C2 I a 2 b	Select * from T;	C1 C2 I a 2 b
C1 C2 I a 2 b	Select C1 from T;	C1 I 2
C1 C2 I a 2 b	Select * from T where C1=I;	C1 C2 I a

6.1. Subconjuntos do SQL

A linguagem SQL é dividida em subconjuntos de acordo com as operações que queremos efetuar sobre um banco de dados, tais como:

DML - Linguagem de Manipulação de Dados.

O primeiro grupo é a DML (Data Manipulation Language - Linguagem de manipulação de dados). DML é um subconjunto da linguagem SQL que é utilizado para realizar inclusões, consultas, alterações e exclusões de dados presentes em registros. Estas tarefas podem ser executadas em vários registros de diversas tabelas ao mesmo tempo. Os comandos que realizam respectivamente as funções acima referidas são INSERT, UPDATE e DELETE.

função	comandos SQL	descrição do comando	exemplo
inclusões	INSERT	é usada para inserir um registro (formalmente uma tupla) a uma tabela existente.	INSERT INTO Pessoa (id, nome, sexo) VALUE;
alterações	UPDATE	para mudar os valores de dados em uma ou mais linhas da tabela existente.	UPDATE Pessoa SET data_nascimento = '11/09/1985' WHERE id_pessoa = 7;
exclusões	DELETE	permite remover linhas existentes de uma tabela.	DELETE FROM pessoa WHERE id_pessoa = 7;

É possível inserir dados na tabela Area usando o INSERT INTO:

```
INSERT INTO Area (arecod, aredes) VALUES (100, "Informática"), (200, "Turismo"), (300, "Higiene e Beleza");
```

DDL - Linguagem de Definição de Dados

O segundo grupo é a DDL (Data Definition Language - Linguagem de Definição de Dados). Uma DDL permite ao utilizador definir tabelas novas e elementos associados. A maioria dos bancos de dados de SQL comerciais tem extensões proprietárias no DDL.

Os comandos básicos da DDL são poucos:

- CREATE: cria um objeto (uma tabela, por exemplo) dentro da base de dados.
- DROP: apaga um objeto do banco de dados.

Alguns sistemas de banco de dados usam o comando ALTER, que permite ao usuário alterar um objeto, por exemplo, adicionando uma coluna a uma tabela existente.

Outros comandos *DDL*:

- CREATE TABLE
- CREATE INDEX
- CREATE VIEW
- ALTER TABLE
- ALTER INDEX
- DROP INDEX
- DROP VIEW

DCL - Linguagem de Controle de Dados

O terceiro grupo é o DCL (Data Control Language - Linguagem de Controle de Dados). DCL controla os aspectos de autorização de dados e licenças de usuários para controlar quem tem acesso para ver ou manipular dados dentro do banco de dados.

Duas palavras-chaves da DCL:

- GRANT - autoriza ao usuário executar ou setar operações.

- REVOKE - remove ou restringe a capacidade de um usuário de executar operações.

DTL - Linguagem de Transação de Dados

- BEGIN WORK - (ou **BEGIN TRANSACTION**, dependendo do dialeto SQL) - pode ser usado para marcar o começo de uma transação de banco de dados que pode ser completada ou não.

- COMMIT - finaliza uma transação dentro de um sistema de gerenciamento de banco de dados.

- ROLLBACK - faz com que as mudanças nos dados existentes desde o último COMMIT ou ROLLBACK sejam descartadas.

COMMIT e ROLLBACK interagem com áreas de controle como transação e locação. Ambos terminam qualquer transação aberta e liberam qualquer cadeado ligado a dados. Na ausência de um BEGIN WORK ou uma declaração semelhante, a semântica de SQL é dependente da implementação.

DQL - Linguagem de Consulta de Dados

Embora tenha apenas um comando, a DQL é a parte da SQL mais utilizada. O comando SELECT permite ao usuário especificar uma consulta ("query") como uma descrição do resultado desejado. Esse comando é composto de várias cláusulas e opções, possibilitando elaborar consultas das mais simples às mais elaboradas.

Função	Comandos SQL	Descrição do comando	Exemplo
consultas	SELECT	O Select é o principal comando usado em SQL para realizar consultas a dados pertencentes a uma tabela.	Select * From Pessoa;

6.2. Palavras-chave em SQL

Cláusulas

As cláusulas são condições de modificação utilizadas para definir os dados que deseja selecionar ou modificar em uma consulta:

- FROM – Utilizada para especificar a tabela, que se vai selecionar os registros.
- WHERE – Utilizada para especificar as condições que devem reunir os registros que serão selecionados.
- GROUP BY – Utilizada para separar os registros selecionados em grupos específicos.
- HAVING – Utilizada para expressar a condição que deve satisfazer cada grupo.
- ORDER BY – Utilizada para ordenar os registros selecionados com uma ordem especifica.
- DISTINCT – Utilizada para selecionar dados sem repetição.
- UNION – combina os resultados de duas consultas SQL em uma única tabela para todas as linhas correspondentes.

6.3. Operadores Lógicos

- AND – E lógico. Avalia as condições e devolve um valor verdadeiro caso ambos sejam corretos.
- OR – OU lógico. Avalia as condições e devolve um valor verdadeiro se algum for correto.
- NOT – Negação lógica. Devolve o valor contrário da expressão.

Operadores relacionais

O SQL possui operadores relacionais, que são usados para realizar comparações entre valores, em estruturas de controle.

Operador	Descrição	Exemplos	
<	Menor	SELECT * FROM informacao.tabela WHERE idade < 18;	Seleciona todos os registros na "tabela" que possuem o campo "idade" com valores **menores** que 18.

>	Maior	SELECT * FROM informacao.tabela WHERE idade > 18;	Seleciona todos os registros na "tabela" que possuem o campo "idade" com valores **maiores** que 18.
<=	Menor ou igual	SELECT * FROM informacao.tabela WHERE idade <= 18;	Seleciona todos os registros na "tabela" que possuem o campo "idade" com valores **menores ou iguais à** 18.
>=	Maior ou igual	SELECT * FROM informacao.tabela WHERE idade >= 18;	Seleciona todos os registros na "tabela" que possuem o campo "idade" com valores **maiores ou iguais à** 18.
=	Igual	SELECT * FROM informacao.tabela WHERE idade = 18;	Seleciona todos os registros na "tabela" que possuem o campo "idade" com valores **exatamente iguais à** 18.
<>	Diferente	SELECT * FROM informacao.tabela WHERE idade <> 18;	Seleciona todos os registros na "tabela" que possuem o campo "idade" com valores que **são diferentes de** 18.

- BETWEEN – Utilizado para especificar valores dentro de um intervalo fechado.
- LIKE – Utilizado na comparação de um modelo e para especificar registros de um banco de dados. "Like" + extensão % significa buscar todos resultados com o mesmo início da extensão.
- IN - Utilizado para verificar se o valor procurado está dentro de um« »a lista. Ex.: valor IN (1,2,3,4).

6.4. Funções de Agregação

As funções de agregação, como os exemplos abaixo, são usadas dentro de uma cláusula SELECT em grupos de registros para devolver um único valor que se aplica a um grupo de registros:

- AVG – Utilizada para calcular a média dos valores de um campo determinado.

- COUNT – Utilizada para devolver o número de registros da seleção.
- SUM – Utilizada para devolver a soma de todos os valores de um campo determinado.
- MAX – Utilizada para devolver o valor mais alto de um campo especificado.
- MIN – Utilizada para devolver o valor mais baixo de um campo especificado.
- STDDEV - Utilizada para funções estatísticas de desvio padrão
- VARIANCE - Utilizada para funções estatísticas de variância

MUITO OBRIGADO!!!