

Linguagem SQL-ANSI, T-SQL e PL-SQL: Subconjunto da Linguagem (DML, DDL, DCL, DTL)**O que é o SQL?**

O SQL é uma linguagem de programação que foi criada na década de 70 pela IBM para os Banco de Dados Relacionais da empresa, com o objetivo de implementar as regras de relacionamento de um banco de dados.

Mais tarde ela entrou para o padrão ANSI e todas as empresas que desenvolvem SGDBs adotaram ele como padrão de banco de dados relacionais.

O que é o PL/SQL?

O PL/SQL é uma linguagem de programação procedural que foi implantada pela Oracle Database no ano de 1991, Clique aqui para ver a história do PL/SQL

Qual a Grande Vantagem de Usar o PL/SQL?

O **PL/SQL** é uma linguagem de programação que é compilada dentro do Banco de Dados e com isso temos uma economia gigantesca de tempo e o poder da utilização do Hardware do Database para processar as informações, esta é uma grande vantagem tendo em vista que o Banco de Dados possui recursos muito poderosos.

Vale lembrar que esses recursos devem ser usados com moderação, como eu costumo defender para os meus alunos o Banco de Dados é o coração da empresa e se ele ficar fora do ar, simplesmente todos os sistemas ficam fora do ar. Então tome muito cuidado com as suas implementações.

É Possível Fazer um Sistema Apenas com o SQL?

A resposta é sim, e eu como especialista nas duas linguagens te digo que qualquer programador deve saber SQL, afinal você vai precisar sempre fazer a interface com um Banco de Dados, inclusive criei um artigo com os 10 motivos pelos quais todo programador deve saber SQL(<http://aprendaplsql.com/2016/03/10-motivos-pelos-quais-todo-programador-deve-saber-sql/>).

Um bom desenvolvedor PL/SQL deve dominar o SQL, isso porque apesar do PL/SQL ser muito rápido e resolver os seus problemas quando um código simples não funciona, o SQL sempre tem que estar muito bem feito, senão você vai ter problemas de performance.

O que é SQL e o que é PL/SQL?**Segue abaixo o que o SQL suporta**

Elementos da linguagem: Clausulas, Expressões, Consultas, Statments (Demonstrações), Condições de uma consulta;

Operadores: =, >, =, <=, BETWEEN, LIKE, IN, NOT IN, IS, IS NOT, AS, etc;

Consultas (Queries): que inclui SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, etc;

Tipos de Dados: NUMBER, CHAR, BIT, DATE e TIME

DDLs(CREATE, REPLACE), DMLs(INSERT, DELETE, UPDATE) e DCLs (COMMIT ROLLBACK)

Já o PL/SQL suporta os seguintes objetos/comandos.

Objetos: Packages, Procedures, Functions, Triggers, Types

Variáveis: NUMBER, PLS_INTEGER, BOOLEAN, BINARY_INTEGER, VARCHAR2, RAW, etc

Estruturas de Decisão e Repetição: IF, ELSE, THEN, LOOP, WHILE, CASE

Cursors Explicitos e Implícitos.

QL é a linguagem de consulta estruturada. SQL é usado para manipular o banco de dados de uma forma simplificada.

PL / SQL é a linguagem processual contém vários tipos de variáveis, funções e procedimentos.

SQL permite que desenvolvedores de emitir única consulta ou executar único insert / update / delete de uma vez, enquanto PL / SQL permite escrever programa completo de ter feito vários insert / update / delete de cada vez.

SQL é uma simples linguagem para manipular dados enquanto linguagem de programação PL / SQL é bem mais complexa.

1. O que é PL/SQL?

Foi introduzida no ano de 1988 como parte do conjunto de tecnologias que compunha a versão 6.0 do SGBD Oracle. Ela possibilita o desenvolvimento de programas que são **armazenados, compilados e executados** dentro do servidor de banco de dados Oracle. É tipicamente utilizada para a criação de aplicações de **missão crítica**, que requerem **alto desempenho** na execução de suas tarefas.

2. Quando usar PL/SQL?

Se a sua empresa trabalha com o SGBD Oracle, a principal vantagem de você criar programas em PL/SQL é, sem dúvida, o fato de a linguagem tornar possível a construção de **aplicações eficientes para a manipulação grandes volumes de dados** (tabelas com milhões ou bilhões de registros). Como o programa PL/SQL é executado dentro do Oracle, os dados manipulados **não** precisam entrar ou sair do SGBD, ou seja, trafegar pela rede. A eficiência da PL/SQL também é garantida através da sua forte integração com a linguagem SQL no ambiente Oracle. É possível executar comandos SQL diretamente de um programa PL/SQL, sem a necessidade da utilização de API's intermediárias (como ODBC ou JDBC).

Como uma segunda característica muito positiva, pode-se dizer que a PL/SQL é significativamente **mais confiável** do que a maioria das outras linguagens de programação. Normalmente, um programa escrito em PL/SQL apresentará um comportamento previsível durante a sua execução. Ele rodará com o desempenho esperado pelo programador e **sem a ocorrência de "bugs inexplicáveis"** tão comuns nos dias de hoje.

O terceiro aspecto positivo é o "tempo de vida" dos programas PL/SQL. Um código escrito em PL/SQL costuma ser mais **durável**, no sentido de que não precisa sofrer manutenção mesmo quando a versão do SGBD é atualizada (ex: mudança da versão Oracle 10g no Windows para Oracle 11g no Linux). É comum encontrar programas PL/SQL que foram escritos há 10 ou mais anos em operação nas empresas. Isto ocorre porque as diferentes versões do PL/SQL são, na maioria dos aspectos, compatíveis.

3. Qual a Diferença entre SQL e PL/SQL?

Uma dúvida frequente entre os iniciantes no mundo Oracle é a seguinte: "afinal de contas, qual a diferença entre SQL e PL/SQL?". A resposta não é difícil. Como o seu próprio nome revela, a **PL/SQL** (Procedural Language extensions to SQL) consiste em uma **extensão** da linguagem **SQL** (Structured Query Language). As características-chave das duas linguagens são descritas a seguir.

SQL

SQL é a linguagem **padrão ANSI** para a manipulação de bancos de dados relacionais. Por ser um padrão aceito pela indústria, é suportada por todos os SGBD's relacionais - o que inclui produtos como Oracle, Microsoft SQL Server, MySQL, PostgreSQL, SQLite e IBM DB2.

Embora seja uma linguagem muito poderosa, o escopo da SQL é claro é direto: oferecer instruções para a **recuperação e manipulação** de dados em tabelas, controle de **transações, definição** de objetos e **controle de acesso**. A tabela abaixo apresenta as principais instruções SQL e suas respectivas funções.

A SQL possui a limitação de ser uma linguagem **declarativa**. Isto significa que não é possível criar um programa inteiro em SQL, pois a linguagem não possui comandos para tomada de decisão (ex: IF-ELSE) e nem para execução de laços (ex: WHILE e FOR).

PL/SQL

A PL/SQL pode ser entendida como uma extensão da linguagem SQL, adicionada de funcionalidades que a tornam uma linguagem de programação completa: controle de fluxo, tratamento de exceções, orientação a objetos, entre outras. Com a PL/SQL podemos escrever **programas inteiros**, desde os mais simples até os mais sofisticados. A linguagem foi criada exatamente com o propósito de oferecer uma solução de programação para os usuários que precisavam escrever aplicações de missão crítica executadas no SGBD Oracle.

No entanto, é importante deixar claro que a PL/SQL não foi criada para ser uma linguagem padrão e nem independente! Ao contrário disso, é uma tecnologia exclusiva Oracle, uma **linguagem proprietária** que pode ser utilizada apenas nos produtos desta empresa. Mas apesar de ser “somente” a “linguagem do banco de dados Oracle” - e não uma linguagem de propósito geral, como Java ou C - a PL/SQL se tornou extremamente difundida ao longo dos anos, possuindo uma enorme comunidade de usuários. Cada nova versão do SGBD Oracle traz embutida uma versão correspondente da PL/SQL incrementada com uma série de novos comandos e funcionalidades.

O quadro mostrado na tabela abaixo apresenta um resumo das diferenças entre SQL e PL/SQL.

Muita gente quando começa a pensar em Banco de dados se depara com algumas definições aos quais ficam bem na dúvida.

Uma das primeiras que sempre levantam, são a diferença entre T/SQL e PL/SQL.

Vale ressaltar que todas tem diversos comandos em comum, porém, cada linguagem é desenvolvida para capacitar alguns recursos específicos desenvolvido por suas marcas.

T/SQL – Transact SQL é a extensão da linguagem SQL lançada pela Microsoft para ser usada pelo SQL Server. Tem como características as variações de suporte ao processamento de Strings, Datas, Functions, etc. Tem modo ágil de declaração de comandos DML, principalmente UPDATE e DELETE e apresenta diversas Variáveis Locais. Costumo dizer que o modo T/SQL é um modo mais limpo para desenvolvimento.

PL/SQL – Procedural Language SQL é a extensão da linguagem SQL lançada pela SUN para ser usada pelo Oracle. É a linguagem que tem como principal característica trabalhar com seus comandos como blocos, sendo processados individualmente. Tem como estrutura básica o seguinte modo: DECLARE > SELECTION > BEGIN > EXCEPTION > END

São os principais métodos. Você vai reparar que o MySQL e sua estrutura é mais voltada ao T/SQL e o DB2 é semelhante ao PL/SQL.

Há sim grande diferença nos métodos de trabalhos, cabe ao usuário, desenvolvedor, etc, ter atenção para saber criar as estruturas corretas e escolher onde se especializar. Lembrando que, ao se especializar em um SGBD, você estará se especializando em seu método de desenvolvimento. Teoria é fácil, aplicar na prática conhecimento em todos os sistemas é outra coisa.

O que é SQL?

Resumindo, a Structured Query Language, mais conhecida pela sigla SQL, é uma linguagem que foi desenvolvida no início dos anos 70, pela IBM, para manipular bancos de dados relacionais. A partir de então, diversos fabricantes de Sistemas Gerenciadores de Bancos de Dados relacionais (SGBDRs), como por exemplo a Oracle, começaram a desenvolver versões próprias da linguagem SQL (chamadas de dialetos ou extensões), e isso levou a necessidade da criação de uma linguagem SQL padronizada.

Com a sua popularização e sucesso, organizações como o **Instituto Americano Nacional de Padrões** (ANSI) e a Organização Internacional de Padronização (**ISO**), resolveram padronizar a linguagem SQL. Em 1986 foi criado um padrão ANSI e em 1987 foi criado um padrão ISO. A partir de

então, surgiram várias versões do padrão SQL, onde cada versão acrescenta novos comandos ou funcionalidades.

História do Padrão ANSI

As versões do padrão ANSI existentes até a presente data são:

-SQL-86: Primeira versão da linguagem, lançada em 1986, consiste basicamente na versão inicial da linguagem criada pela IBM.

-SQL-92: Lançada em 1992, inclui novos recursos tais como tabelas temporárias, novas funções, expressões nomeadas, valores únicos etc.

-SQL:1999 (SQL3): Lançada em 1999, foi a versão que teve mais recursos novos significativos, entre eles: a implementação de expressões regulares, recursos de orientação a objetos, queries recursivas, triggers, novos tipos de dados (boolean, LOB, array e outros), novos predicados etc.

-SQL:2003: Lançada em 2003, inclui suporte básico ao padrão XML, sequências padronizadas, instrução MERGE, colunas com valores auto-incrementais etc.

-SQL:2006: Lançada em 2006, não inclui mudanças significativas para as funções e comandos SQL. Contempla basicamente a interação entre SQL e XML.

Padrão Oracle X padrão ANSI

Atualmente, os principais fabricantes de SGBDRs, implementam em seus Bancos de Dados, além das instruções SQL referentes ao seu "dialeto", as instruções SQL do padrão ANSI mais recente. No caso de instruções SQL no SGBD Oracle, o que o pessoal costuma chamar de **padrão Oracle**, é o dialeto SQL da Oracle. As instruções dos dialetos normalmente surgem quando o fabricante necessita implementar recursos no SGBD, que ainda não possuem instruções correspondentes no padrão ANSI.

Agora que já sabemos o que é o padrão Oracle e o que é o padrão ANSI, vou explicar qual desses padrões pode proporcionar melhor performance às instruções SQL, no caso do Banco de Dados Oracle. Antes ainda, de falar sobre a performance dos 2 padrões, vou comentar sobre outros itens, que me fazem defender o uso do **padrão ANSI**. Entre eles, vou destacar os seguintes:

- Permite que você migre a aplicação contendo as instruções SQL para outro BD sem ter que fazer qualquer alteração;

- As ligações entre colunas são mais fáceis de ser identificadas, pois ficam fora da cláusula WHERE (separadas das condições de filtro). Essa facilidade pode muitas vezes facilitar manutenção futura e evitar queries ruins. Já vi muita gente escrever queries no padrão Oracle e esquecer de incluir uma coluna necessária para a ligação correta entre 2 tabelas. Para evitar linhas duplicadas, resultantes do produto cartesiano gerado pela falta da coluna necessária na ligação, o desenvolvedor incluía a instrução DISTINCT. "Uma tremenda gambiarra para corrigir um erro de programação, que gerava degradação da performance da query";

- Permite fazer OUTER JOIN com parâmetros de stored procedures. No padrão Oracle você não consegue fazer isso. Eu comecei a escrever instruções SQL no padrão ANSI em 2005, quando percebi que somente o padrão ANSI permitia isso;

Bom... agora quanto à performance: utilize o padrão ANSI! Desde a implantação do **otimizador baseado em custo** (CBO) no Oracle Database, a Oracle recomenda a utilização do padrão ANSI, pois o CBO foi desenvolvido com base neste padrão. Deste modo, as instruções SQL escritas no padrão ANSI podem gerar planos de execução melhores, e consequentemente, serão executadas mais rapidamente!

Na prática, dificilmente você encontrará diferenças no plano de execução gerado entre os 2 padrões, mas... um dia você encontrará! Isso já aconteceu comigo em uma turma de SQL Tuning que lecionei em 08/2011. Uma das alunas refez uma instrução SQL do treinamento utilizando o padrão Oracle e o plano de execução gerado pelo otimizador ficou diferente. No plano da instrução ANSI o otimizador

usava um índice para acessar uma das tabelas. No plano da instrução Oracle o otimizador fazia Full Table Scan, logo, a instrução SQL escrita no padrão ANSI tinha um tempo de execução melhor!

O que é SQL?

SQL (Structured Query Language) foi desenvolvido originalmente no início dos anos 70, a sigla significa em português Linguagem de Consulta Estruturada. Ela é a linguagem padrão dos SGDBs e segue o padrão ANSI na sua essência, tendo poucas variações de SGDBs para SGDBs

A linguagem SQL é independente de hardware ou de software, ou seja, o usuário não precisa saber a respeito de bancos de dados(SGDBs) envolvidos na operação ou sobre o funcionamento de quem vai executar a instrução. Se você quer ser um programador, prepare-se: você vai precisar aprender SQL. Clique Abaixo e veja o vídeo onde eu explico em detalhes o SQL e suas divisões

História do SQL?

O SQL tornou-se popular por ser um programa simples e de fácil uso, assim sendo, cada busca de dados resultada na especificidade do que o usuário procura e não no caminho para chegar à ele.

Criado pela IBM, logo ele foi sendo redefinido por outros produtores, o que levou à uma expansão da necessidade de uma linguagem padronizada. E foi justamente que a American National Standards Institute (ANSI) fez.

Depois, em cada revisão o programa ganharia um subtítulo, de acordo com o ano, tais como SQL 92, SQL 1999 e SQL 3. E embora seja padronizado pela ANSI e ISO, ele possui muitas variações e extensões produzidas por diferentes fabricantes, que trazem variações nas estruturas principais.

Como o SQL Funciona

O SQL é dividido em subconjuntos que variam conforme as operações que o usuário efetua no banco de dados. Listamos os subconjuntos, leiam:

1 – DML (Linguagem de Manipulação de Dados)

É utilizado para realizar inclusões, exclusões e alterações de dados presentes em registros. Aqui, os principais comandos são Insert, Update e Delete. Alguns não concordam com a existência do DQL, que vocês verão logo abaixo, e incluirão o SELECT na lista de comandos dos DMLs

2 – DDL (Linguagem de Definição de Dados)

Permite ao usuário definir tabelas e elementos associados. Aqui, os comandos são CREATE e DROP, um cria o objeto dentro da base e o outro apaga um objeto do banco de dados. Algumas exceções usam também o ALTER, que permite adicionar coluna à tabela existente.

3 – DCL (Linguagem de Controle de Dados)

É o controlador das autorizações de dados e licenças para o usuário que for controlar ou manipular os dados do banco. GRANT e REVOKE são as chaves, uma autoriza a execução das operações e a outra remove ou restringe.

4 – DTL (Linguagem de Transação de Dados)

Aqui acontece a interação entre as áreas de controle, tais como transação e locação. Os comandos usados são BEGIN WORK, que pode ser usado para marcar o começo da transação; COMMIT, que finaliza a transação e ROLLBACK que faz as mudanças nos dados existentes.

5 – DQL (Linguagem de Consulta de Dados)

Essa é a parte mais utilizada, que tem um único comando, o SELECT, que permite especificar onde uma consulta como a descrição do resultado desejado.

Como usar o SQL

No SQL você precisa conhecer inglês, pelo menos do ponto de vista técnico, esta é para mim uma premissa para você ser um bom profissional de TI. Abaixo, vamos indicar as 3 principais palavras usadas nessa programação e vamos defini-la, mas, atenção, essa é apenas uma degustação, pois existem outras que também devem estar na “ponta da língua” de qualquer DBA. Se você quer ser um bom DBA leia este artigo!

- **Select:** usado para acessar informações no banco de dados;
- **From:** usado para indicar de que tabela de dados as informações serão obtidas;
- **Where** nome: comando opcional usado para restringir os resultados.

Claro que antes de usar essas palavras-chaves será preciso criar as tabelas e inserir os registros. E, deve-se saber que o idioma não deve ser um empecilho, mesmo porque a linguagem SQL é um dos maiores sucesso da área de TI, sendo ainda mais notado quando agiliza todo o processo que poderia demorar horas ou dias.

O que são Schemas?

Uma dúvida recorrente entre os profissionais é sobre os Schemas, que são coleções de objetos dentro do banco de dados. Eles se organizam e são importantes para a segurança do programa, facilitando a administração dos objetos e dos dados. Eles são a ponta de associação entre o usuário e o objeto do banco.

Eles são recomendados em bases de dados de múltiplos bancos de dados, na qual é necessária a autorização ou revogação dos usuários e grupos.

Comandos DDL, DQL, DML, DCL, DTL, TCL?

Os comandos SQL são agrupados em cinco categorias. Segue abaixo suas definições e utilizações.

DDL – Data Definition Language - Linguagem de Definição de Dados.

Estes comandos são utilizados para definir a estrutura de banco de dados, criando ou removendo objetos.

CREATE- criar banco de dados, tabelas, colunas.

DROP - remover um objeto no banco de dados.

ALTER – altera a estrutura da base de dados

TRUNCATE – remover todos os registros de uma tabela, incluindo todos os espaços alocados para os registros são removidos. Limpa a tabela por completo. Semelhante ao parâmetro Purge de remoção de programas no Linux.

COMMENT – adicionar comentários ao dicionário de dados

RENAME – para renomear um objeto

DQL – Data Query Language - Linguagem de Consulta de Dados.

Utilizado para consultas dos dados.

SELECT- recuperar dados do banco de dados

DML – Data Manipulation Language - Linguagem de Manipulação de Dados.

Utilizados para o gerenciamento de dados dentro de objetos do banco.

INSERT – inserir dados em uma tabela

UPDATE – atualiza os dados existentes em uma tabela

DELETE – exclui registros de uma tabela,

CALL – chamar um subprograma PL / SQL

EXPLAIN PLAN – explicar o caminho de acesso aos dados

LOCK TABLE – controle de concorrência

DCL – Data Control Language - Linguagem de Controle de Dados

Conjunto de comandos utilizados para controlar o nível de acesso de usuários.

GRANT – atribui privilégios de acesso do usuário a objetos do banco de dados

REVOKE – remove os privilégios de acesso aos objetos obtidos com o comando GRANT

DTL OU TCL – Transaction Control Language - Linguagem de Transação de Dados

São usados para gerenciar as mudanças feitas por instruções DML . Ele permite que as declarações sejam agrupadas em transações lógicas .

COMMIT – salvar o trabalho feito

SAVEPOINT – identificar um ponto em uma transação para que mais tarde você pode efetuar um ROLLBACK.

ROLLBACK – restaurar banco de dados ao original desde o último COMMIT

Estes comandos são os principais usados no gerenciamento, manutenção e consulta de um banco de dados relacional.

DDL – Data Definition Language (DDL) são usadas para definir a estrutura de banco de dados ou esquema. Alguns exemplos:

CREATE- para criar objetos no banco de dados

ALTER – altera a estrutura da base de dados

TRUNCATE – remover todos os registros de uma tabela, incluindo todos os espaços alocados para os registros são removidos

COMMENT – adicionar comentários ao dicionário de dados

RENAME – para renomear um objeto

DML – Data Manipulation Language (DML) são utilizados para o gerenciamento de dados dentro de objetos do banco. Alguns exemplos:

SELECT- recuperar dados do banco de dados

INSERT – inserir dados em uma tabela

UPDATE – atualiza os dados existentes em uma tabela

DELETE – exclui registros de uma tabela,

CALL – chamar um subprograma PL / SQL

EXPLAIN PLAN – explicar o caminho de acesso aos dados

LOCK TABLE – controle de concorrência

DCL – Data Control Language (DCL) declarações. Alguns exemplos:

GRANT – atribui privilégios de acesso do usuário a objetos do banco de dados

REVOKE – remove os privilégios de acesso aos objetos obtidos com o comando GRANT

TCL – Transaction Control Language – (Controle de Transações) são usados para gerenciar as mudanças feitas por instruções DML . Ele permite que as declarações a serem agrupadas em transações lógicas .

COMMIT – salvar o trabalho feito

SAVEPOINT – identificar um ponto em uma transação para que mais tarde você pode efetuar um ROLLBACK

ROLLBACK – restaurar banco de dados ao original desde o último COMMIT

A linguagem Transact-SQL é uma extensão ao padrão SQL-92, sendo a linguagem utilizada por desenvolvedores na construção de aplicações que manipulam dados mantidos no SQL Server. Seus comandos podem ser classificados em quatro grupos, de acordo com sua função, em: DML (Linguagem de Manipulação de Dados), DDL (Linguagem de Definição de Dados), DCL (Linguagem de Controle de Dados) e DTL (Linguagem de Transação de Dados).

Além dessas categorias, podemos ter também uma relacionada à consulta dos dados (DQL – Linguagem de Consulta de Dados), que possui apenas o comando SELECT. Entretanto, é mais comum encontrar esse comando como parte da DML em conjunto com os demais comandos de manipulação INSERT, UPDATE e DELETE.

DDL

Esse subconjunto apoia a criação de objetos no banco de dados, alterar a estrutura da base de dados ou deletar o banco de dados. Seus principais comandos são:

- CREATE;
- ALTER;
- DROP.

DML

Esse subconjunto é utilizado para realizar consultas, inclusões, alterações e exclusões de dados. Seus principais comandos são:

- SELECT ... FROM ... WHERE ...
- INSERT INTO ... VALUES ...
- UPDATE ... SET ... WHERE ...
- DELETE FROM ... WHERE ...

DCL

Esse subconjunto da linguagem SQL é responsável por controlar os aspectos de autorização de dados e a utilização de licenças por usuários. São conhecidos como comandos DCL (Data Control Language):

- GRANT: comando usado para fornecer acesso ou privilégios sobre os objetos de banco de dados para os usuários;
- DENY: nega a permissão a um usuário ou grupo para realizar uma determinada operação em um objeto ou recurso;
- REVOKE: remove a permissão GRANT ou DENY.

DTL

Esse subconjunto da linguagem SQL é responsável por gerenciar transações executadas no banco. Seus principais comandos são:

- BEGIN TRAN (OU BEGIN TRANSACTION): marca o começo de uma transação no banco de dados que pode ser completada ou não;
- COMMIT: efetiva e envia os dados de forma permanente para o banco de dados;
- ROLLBACK: retorna o banco ao estado anterior, desfazendo as alterações feitas na transação.

Integridade de Dados

Outro conceito essencial e presente nos SGBDs é o de integridade de dados. Ele diz respeito a uma série de restrições impostas pelos SGBDs para garantir que os dados nunca saiam de um estado consistente para um inconsistente. Existem diferentes tipos de mecanismos de integridade implementados pelos SGBDs. A integridade de entidade garante que cada linha em uma tabela é um registro exclusivamente identificável. Você pode aplicar a integridade de entidade para uma tabela

especificando uma restrição PRIMARY KEY. Por exemplo, a coluna ProductID da tabela produtos é uma chave primária para a tabela.

A integridade referencial, definida através da restrição FOREIGN KEY, assegura que as relações entre tabelas permanecem preservadas ao longo do tempo. Já a integridade de domínio garante que os valores de dados dentro de um banco seguem regras definidas para valores, alcance e formato. Um banco de dados pode impor essas regras usando uma variedade de técnicas, incluindo restrições CHECK, restrições UNIQUE e restrições padrão. Essas serão as restrições apresentadas nesse artigo, mas precisamos estar cientes de que existem outras opções disponíveis para impor a integridade de domínio.

A lista a seguir fornece alguns exemplos de restrições de integridade de domínio:

- Um nome do produto não pode ser NULL;
- Um nome do produto deve ser exclusivo;
- A data de uma ordem não deve ser no futuro;
- A quantidade de produto em uma ordem deve ser maior do que zero.

Tipos de dados no SQL Server

Há diferentes tipos de dados no SQL Server: de sistema e de usuários (User-Defined Types-UDTs) ou SQL Common Language Runtime (CLR). A seguir temos uma amostra dos tipos disponíveis no SQL Server:

String:

- CHAR;
- VARCHAR;
- NCHAR;
- NVARCHAR;
- TEXT;
- NTEXT.

O que significa a sigla SQL? Qual a finalidade dessa linguagem? Como ela se divide? Quais são os comandos principais de cada divisão? Explique-os resumidamente.

Structured Query Language, ou Linguagem de Consulta Estruturada ou SQL, é uma linguagem de pesquisa declarativa para banco de dados relacional. Surgiu para padronizar a manipulação dos dados do BD. É dividida em:

DML - Linguagem de Manipulação de Dados

Primeiro há os elementos da DML (Data Manipulation Language - Linguagem de Manipulação de Dados). A DML é um subconjunto da linguagem usada para inserir, atualizar e apagar dados. INSERT, UPDATE, DELETE

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. CREATE, DROP

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.
GRANT, REVOKE

DTL - Linguagem de Transação de Dados

BEGIN WORK, COMMIT, ROLLBACK

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.
FROM, WHERE, GROUP BY, HAVING, ORDER BY, DISTINCT.

2) Quais são os tipos de dados que a linguagem SQL padrão (ANSI) suporta? Explique cada um resumidamente.

Os dados podem ser armazenados em diferentes tipos, o tipo define as operações que poder ser executadas com determinado dado. Os tipos no SQL são:

Inteiros: integer, int, smallint e tinyint;
Reais: float, double, real, numeric;
Caracteres: char;
Texto: varchar, text;
Data: date

3) Explique o comando SELECT. Sua sintaxe, características básicas, Cláusula Where, operadores, conectores, subconsultas, funções de agregação, cláusula order by, group by, having e compute by.

O comando **SELECT** Instrui o programa principal do banco de dados para retornar a informação como um conjunto de registros.
Para executar esta operação, o programa principal de banco de dados procura a tabela ou tabelas especificadas, extrai as colunas escolhidas, seleciona as linhas que satisfazem o critério e classifica ou agrupa as linhas resultantes na ordem especificada.

Sintaxe

```
SELECT [predicado { * tabela.* [tabela.]campo1 [AS alias1] [, [tabela.]campo2 [AS alias2] [, ...]]}  
FROM expressão[tabela [, ...] [IN bancodedadosexterno]  
[WHERE... ]  
[GROUP BY... ]  
[HAVING... ]  
[ORDER BY... ]  
[WITH OWNERACCESS OPTION]
```

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 específica.

Funções de Agregação.

As funções de soma se usam 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 media 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.

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

< – Menor que

> – Maior que

<> – Diferente de

<= – Menor ou Igual que

>= – Maior ou Igual que

= – Igual a

BETWEEN – Utilizado para especificar um intervalo de valores.

LIKE – Utilizado na comparação de um modelo e para especificar registros de um banco de dados. "Like" + extensão % vai significar buscar todos resultados com o mesmo início da extensão.

4) O que é uma visão (view) em SQL? Explique suas propriedades, utilidades e os comandos para a sua manipulação.

Uma view é uma maneira alternativa de observação de dados de uma ou mais entidades (tabelas), que compõem uma base de dados. Pode ser considerada como uma tabela virtual ou uma consulta armazenada. Geralmente é recomendável, uma view, implementada encapsulando uma instrução SELECT (busca de dados para exposição), guarda os dados em uma tabela virtual, armazenando também em cache, pois todas as consultas ao banco, encapsuladas ou não, ao serem executadas, são armazenadas em cache. Por este motivo, pode ser mais rápido ter uma consulta armazenada em forma de view, em vez de ter que retrabalhar uma instrução.

O comando para criar uma view é o **CREATE VIEW**.

5) O que são procedimentos armazenados (stored procedure)? Para que servem, o que eles permite fazer e quais comandos podemos utilizar no seu corpo e para criá-los?

Stored Procedure é um conjunto de comandos, ao qual é atribuído um nome. Este conjunto fica armazenado no Banco de Dados e pode ser chamado a qualquer momento tanto pelo SGBD (sistema Gerenciador de Banco de Dados) quanto por um sistema que faz interface com o mesmo. A utilização de Stored Procedures é uma técnica eficiente de executarmos operações repetitivas. Ao invés de digitar os comandos cada vez que determinada operação necessite ser executada.

6) O que são gatilhos (TRIGGERS)? Para que servem, quais as vantagens na sua utilização, como funcionam e como criá-los?

Um Trigger é bloco de comandos Transact-SQL que é automaticamente executado quando um comando INSERT, DELETE ou UPDATE for executado em uma tabela do banco de dados. Os Triggers são usados para realizar tarefas relacionadas com validações, restrições de acesso,

rotinas de segurança e consistência de dados ; desta forma estes controles deixam de ser executados pela aplicação e passam a ser executados pelos Triggers em determinadas situações :
Mecanismos de validação envolvendo múltiplas tabelas

Criação de conteúdo de uma coluna derivada de outras colunas da tabela
Realizar análise e e atualizações em outras tabelas com base em alterações e/ou inclusões da tabela atual

A criação de um Trigger envolve duas etapas :

Um comando SQL que vai disparar o Trigger (INSERT , DELETE , UPDATE)

A ação que o Trigger vai executar (Geralmente um bloco de códigos SQL)

Podemos criar um Trigger usando o comando **Create Trigger**

Conceitos Básicos de SQL

A linguagem SQL (Structured Query Language) e a linguagem para banco de dados mais utilizada atualmente. Ela pode consultar banco de dados, definir a estrutura dos dados, modificar dados em um banco de dados e especificar restrições de segurança.

A SQL é uma linguagem declarativa, que especifica o resultado da consulta, mas não detalha como a consulta é feita.

Abrange alguns tipos de linguagem de dados:

- **DDL** (Data Definition Language): serve para a definição de estruturas de dados (conceitual), incluindo linhas, colunas, tabelas, índices e localizações de arquivos. Ex:

- **create**: para criar objetos no banco de dados (inclui o create view)

- **alter**: altera a estrutura do banco de dados

- **drop**: apaga objetos do banco de dados

- **truncate**: remove todos os registros de uma tabela, incluindo todos os espaços alocados para os registros são removidos. Não permite rollback.

- **comment**: adicionar comentários ao dicionário de dados

- **rename**: renomear um objeto.

- **DML** (Data Manipulation Language): voltada para a manipulação dos dados como o próprio nome diz. Ex:

- **insert**: inserir dados em uma tabela

- **select**: recuperar dados do banco de dados (pode ser considerada da DQL por algumas bancas)

- **update**: atualiza os dados existentes em uma tabela

- **delete**: exclui registros de uma tabela. Passível de rollback

- **call**: chamar um subprograma

- **explain plan**: explicar o caminho de acesso aos dados

- **lock table**: controle de concorrência

- **TCL** (Transaction Control Language): para a transação no banco de dados. Ex:

- **begin transaction**

- **commit**: salvar o trabalho feito

- **rollback**: restaurar o banco de dados ao original desde o último commit

Bancos de Dados Relacionais

Para início de discussão, um banco de dados relacional é um mecanismo de armazenamento que permite a persistência de dados e opcionalmente implementar funcionalidades. Neste contexto, o objetivo deste artigo é apresentar uma visão geral de tecnologias de sistemas de gerenciamento de banco de dados relacionais (SGBDR) e explorar questões práticas aplicáveis ao seu uso em organizações modernas. Sendo assim, o objetivo deste artigo não é discutir a teoria relacional.

SGBDRs são usados para armazenar a informação requerida por aplicações construídas usando tecnologias procedurais, tais como COBOL ou FORTRAN, tecnologias orientadas a objeto tais como Java e C# e tecnologias baseadas em componentes como Visual Basic. Como SGBDRs são as tecnologias de armazenamento de persistência dominantes, é importante que todos os profissionais de TI entendam ao menos os conceitos básicos dos SGBDRs, os desafios por trás da tecnologia e quando seu uso é apropriado.

Conhecendo Um Sistema Gerenciador De Banco De Dados Relacional

Vamos iniciar este tópico pela definição de algumas terminologias comuns. Um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) é um software que controla o armazenamento, recuperação, exclusão, segurança e integridade dos dados em um banco de dados. Um banco de dados relacional armazena dados em tabelas. Tabelas são organizadas em colunas, e cada coluna armazena um tipo de dados (inteiro, números reais, strings de caracteres, data, etc.). Os dados de uma simples “instância” de uma tabela são armazenados como uma linha. Por exemplo, a tabela Cliente teria colunas como numeroCliente, primeiroNome e sobrenome, e uma linha na tabela teria algo como {123, “Arilo”, “Dias”}.

Tabelas tipicamente possuem chaves, uma ou mais colunas que unicamente identificam uma linha na tabela. No caso da tabela Cliente a chave seria a coluna numeroCliente. Para melhorar o tempo de acesso aos dados de uma tabela, são definidos índices. Um índice provê uma forma rápida para buscar dados em uma ou mais colunas em uma tabela, da mesma forma que o índice de um livro permite que nós encontremos uma informação específica rapidamente.

Funcionalidades Básicas De Sgbdrs

O uso mais comum de SGBDRs é para implementar funcionalidades simples do tipo CRUD (do inglês Create, Read, Update e Delete – que significa as operações de Inserção, Leitura, Atualização e Exclusão de dados). Por exemplo, uma aplicação pode criar uma nova compra e inseri-la no banco de dados. Ela pode ler uma compra, trabalhar com seus dados e então atualizar o banco de dados com a nova informação. Ela pode ainda optar por excluir uma compra existente, talvez porque o cliente a cancelou. A grande maioria das interações com um banco de dados provavelmente implementará as funcionalidades básicas de CRUD.

A forma mais fácil de manipular um banco de dados relacional é submeter declarações escritas na linguagem SQL a ele. A Figura 1 descreve um simples modelo de dados usando a notação de modelagem de dados proposta pela UML. Para criar uma nova linha na tabela Seminario devemos criar uma declaração de INSERT, como exemplificado no código da Listagem 1. Similarmente, o código da Listagem 2 apresenta um exemplo de como ler uma linha da tabela usando a declaração SELECT. A Listagem 3 apresenta um código para atualizar uma linha existente através da declaração do tipo UPDATE e, finalmente, a Listagem 4 descreve como excluir uma linha da tabela usando a declaração DELETE.

Listagem 1. Declaração SQL para inserir uma linha na tabela Seminario

```
INSERT INTO Seminario
```

```
(idSeminario, idCurso, idProfessor, numeroSeminario, tituloSeminario)
```

```
VALUES (74656, 1234, 'DCC1982', 2, "Banco de Dados Relacional")
```

Listagem 2. Declaração SQL para consultar uma linha na tabela Seminario


```
SELECT * FROM Seminario WHERE SEMINAR_ID = 1701
```

Listagem 3. Declaração SQL para atualizar uma linha na tabela **Seminario**

```
UPDATE Seminario
```

```
SET idProfessor = 'PPGI1982', numeroSeminario = 3
```

```
WHERE idSeminario = 1701
```

Listagem 4. Declaração SQL para excluir uma linha na tabela **Seminario**

```
DELETE FROM Seminario
```

```
WHERE idSeminario > 1701 AND idProfessor = 'THX0001138'
```

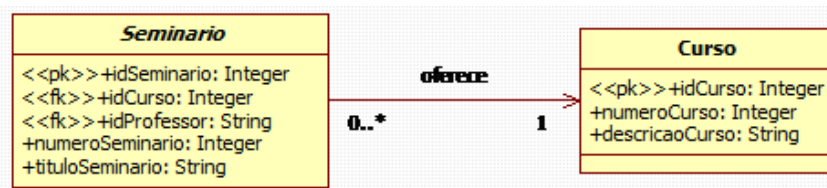


Figura 1. Um modelos de dados simples em notação UML.

Uso avançado de SGBDRs

Existem várias funcionalidades avançadas de SGBDRs que desenvolvedores aprendem uma vez que eles estão familiarizados com as funcionalidades básicas de CRUD. Cada uma dessas funcionalidades é muito importante, e às vezes bastante complexa, fazendo com que tivéssemos que escrever um artigo próprio para cobri-las. Por isso, iremos aqui apenas introduzir os conceitos e então detalhes podem ser encontrados em outros artigos. Essas funcionalidades incluem:

Armazenamento de Objeto

Para armazenar um objeto em um banco de dados relacional precisamos adequá-lo – criar uma representação de dados do objeto em questão – pois bancos de dados relacionais apenas armazenam dados. Para recuperar o objeto, é preciso ler os dados a partir do banco de dados e então criar o objeto, operação normalmente chamada de restauração do objeto, baseado nos dados recuperados. Apesar de o armazenamento em um banco de dados relacional parecer algo simples, na prática não é. Isso porque não existe uma tradução perfeita e automática entre as tecnologias de objeto e relacional, pois essas tecnologias são baseadas em teorias diferentes. Para armazenar objetos com sucesso em bancos de dados relacionais, precisamos aprender como mapear um esquema de objetos para um esquema de banco de dados relacional.

Implementar Comportamento No Banco De Dados

Comportamento é implementado em um banco de dados relacional através de stored procedures e/ou stored functions que podem ser invocadas internamente no banco de dados e por aplicações externas. Stored functions e procedures são operações que executam no SGBDR, a diferença entre elas é o que a operação pode retornar e se ela pode ser invocada em uma query. As diferenças não são importantes para nosso objetivo neste artigo, então usaremos o termo stored procedure para se referir a ambas as operações. No passado, stored procedures eram escritas em uma linguagem proprietária, tal como PL/SQL da Oracle, mas agora Java está se tornando rapidamente uma opção de linguagem para programação de banco de dados. Uma stored procedure tipicamente executa algum código SQL, mensagens de dados e então aguarda uma resposta na forma de zero ou mais registros, um código de resposta ou uma mensagem de erro de banco de dados.

Controle de Concorrência

Considere um sistema de reserva de passagem aéreas. Existe um voo com um assento e duas pessoas estão tentando reservar este assento ao mesmo tempo. Ambas as pessoas verificam o status

do voo e são avisadas que o assento ainda está disponível. Ambos informam seus dados para pagamento do ticket e então clicam no botão de reserva ao mesmo tempo. O que deveria acontecer? Se o sistema está funcionando corretamente, apenas uma pessoa deveria ter acesso ao assento e a outra deveria ser informada que não existe nenhum assento disponível. Controle de concorrência é o que faz isso acontecer. Ele deve ser implementado ao longo do código fonte do objeto no banco de dados.

Controle de Transação

Uma transação é uma coleção de ações no banco de dados – tal como salvar, recuperar ou deletar – que formam uma unidade de trabalho. Uma bandeira das transações é uma abordagem que diz “tudo ou nada”, o que significa que todas as ações devem ser executadas com sucesso ou então serem desfeitas (operação de roll back). A transação aninhada é uma abordagem onde algumas das ações são tratadas como suas próprias transações, subdividindo-as. Essas subtransações são comitadas uma vez com sucesso e não são desfeitas se a transação maior falhar. Transações podem ainda ser de curta duração, executando em centésimos de segundo, ou de longa duração, levando horas, dias, semanas e até meses para serem completadas. Controle de transação é um conceito crítico que todos desenvolvedores devem entender.

Forçar Integridade Referencial

Integridade referencial (IR) é a garantia de que uma referência a partir de uma entidade para outra entidade é válida. Por exemplo, se um cliente referencia um endereço, então este endereço deve existir. Se o endereço é deletado, então todas as referências a ele devem também ser removidas, caso contrário o sistema não deve permitir a operação de exclusão.

Contrariando a crença popular, IR não é apenas uma questão de banco de dados, mas sim um aspecto a ser tratado no sistema como um todo. Um cliente é implementado como um objeto em uma aplicação Java e como um ou mais registros no banco de dados – endereços são também implementados como objetos e como linhas. Para excluir um endereço, devemos remover o objeto endereço da memória, qualquer referência direta ou indireta a ele (uma referência indireta para um endereço incluiria um objeto cliente que conhece o valor de idEndereco, a chave primária de endereço no banco de dados), a(s) linha(s) de endereço no banco de dados e qualquer referência a ela (através de chaves estrangeiras) no banco de dados.

Para complicar ainda mais, se temos outras aplicações acessando o banco de dados, então é possível que elas possuam representações do endereço em memória. Um cenário ainda pior seria se tivéssemos o endereço armazenado em vários locais (ex: bancos de dados diferentes), devemos levar isso em consideração. Todos os desenvolvedores devem entender as estratégias básicas para implementar integridade referencial.

A Tabela 1 descreve as funcionalidades técnicas comuns nos principais SGBDRs disponíveis no mercado, as principais formas dos desenvolvedores usá-los e os pontos negativos associados ao seu uso.

Funcionalidades	Uso Principal	Pontos Negativos
Database cursors– Um database cursor é um objeto usado para percorrer os resultados de uma consulta SQL, permitindo mover para frente ou para trás no conjunto de resultados acessando um ou vários registros por vez.	Acessar um grande conjunto de resultados em porções menores permitindo à aplicação exibir resultados iniciais antes, melhorando o tempo de resposta. Desempenho é melhorado quando uma porção de um conjunto de resultado é requerido porque menos dados são transmitidos pela rede.	Desenvolvedores precisam entender que os dados exibidos podem mudar entre as vezes que os registros de dados são acessados via o cursor: registros previamente retornados podem ter sido excluídos, incluídos ou até mesmo modificados dentro daquele conjunto. Nem todos os cursors são criados igualmente. Alguns permitem apenas mover para frente. Cursors são recursos que consomem muita memória em um SGBDR.

Java – A maioria dos SGBDRs comerciais suportam a máquina virtual de Java no banco de dados.	Desenvolvimento independente de plataforma no banco de dados. Desenvolvimento de sistemas com grande carga de dados que resulta em um baixo valor de retorno. Encapsulamento do acesso ao banco de dados para apoiar controle de acesso seguro às informações. Implementação de comportamento compartilhado requerido por muitas aplicações.	Versão diferente de máquina virtual entre o servidor de aplicação e o de banco de dados aumenta a complexidade do desenvolvimento. Comportamento implementado no banco de dados pode se tornar facilmente um gargalo para a aplicação.
Triggers – Um trigger é um procedimento que é executado antes ou após uma ação (tal como inserção, atualização ou exclusão), e é executado em uma linha de uma tabela do banco de dados.	Garantir a integridade referencial no banco de dados. Esses tipos de triggers podem ser gerados automaticamente pela ferramenta de modelagem dos dados ou de administração de banco de dados. Normalmente um recurso mais simples para implementar restrições de integridade referencial. Realiza auditoria através de arquivos de log das alterações manuais.	Triggers podem ser difíceis de serem mantidas e aumentará a dependência para o fabricante do banco de dados. Triggers são tipicamente implementadas em uma linguagem proprietária, que requer uma habilidade extra da equipe. Como triggers são automaticamente invocadas, elas podem ser muito perigosas (ex: exclusões em cascata “descontroladas” resultante de uma trigger de exclusão). O comportamento implementado no banco de dados pode facilmente se tornar um gargalo se o banco de dados não for bem escalado.

Tabela 1. Funcionalidades Técnicas Comuns de SGBDR.

Acoplamento: seu maior inimigo

Acoplamento é uma medida do nível de dependência entre dois itens – quanto mais acoplado são dois elementos, maior a chance de que uma mudança em um deles irá requerer uma mudança no outro. Acoplamento é a “raiz de todo mal” quando se fala do desenvolvimento de software, e quanto mais coisas seu banco de dados está acoplado mais difícil é mantê-lo e evoluí-lo. Esquemas de banco de dados relacional podem ser acoplados a:

O código de fonte de sua aplicação:

Quando o esquema do banco de dados é alterado o código fonte da aplicação que acessa a parte modificada no esquema também precisa ser alterado. A Figura 2 descreve o cenário do melhor caso – quando o esquema do banco de dados é acoplado apenas ao esquema do banco de dados. Esses cenários existem e normalmente são encontrados em aplicações stand-alone, apesar de serem raros na prática.

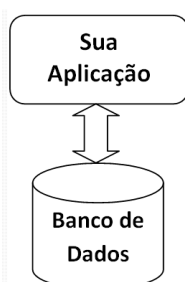


Figura 2. O cenário do melhor caso.

O código fonte de outra aplicação:

A Figura 3 descreve o cenário do pior caso para bancos de dados relacionais – umas grandes variedades de sistemas estão acoplados ao esquema do banco de dados, uma situação que é bastante comum na prática.

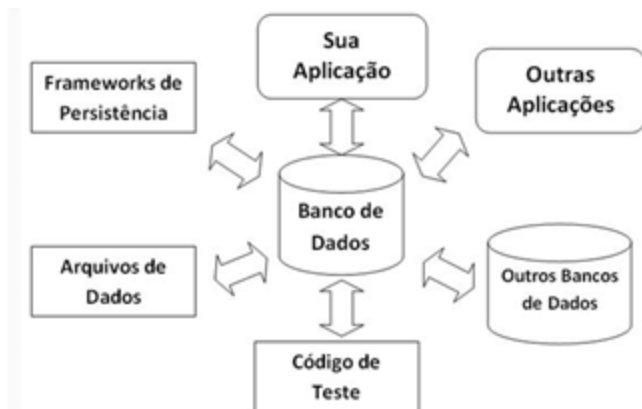


Figura 3. O cenário do pior caso.

O código fonte de carga de dados:

Carga de dados de outras fontes, como tabelas externas ou dados de teste, normalmente são acoplados ao esquema do banco de dados.

O código fonte de extração de dados:

Podem existir scripts ou programas de extração de dados que leem dados a partir do banco de dados, talvez para produzir um arquivo de dados em XML ou simplesmente para que seus dados sejam carregados em outro banco de dados.

Camadas/frameworks de persistência:

Um framework de persistência encapsula a lógica para mapear as classes da aplicação para fontes de armazenamento de persistência, como um banco de dados.

O esquema de banco de dados:

Acoplamento pode existir no banco de dados. Uma simples coluna é acoplada a qualquer stored procedure que a referência, outras tabelas que a usam como chave estrangeira, qualquer view que a referência, dentre outras coisas. Uma simples mudança pode resultar em várias mudanças no banco de dados.

Scripts de migração dos dados:

Mudanças no esquema do banco de dados irão requerer mudanças no script de migração dos dados.

Código de teste:

Código de teste inclui qualquer código fonte que deixe o banco de dados em um estado conhecido, que realiza transações que afetam o banco de dados e que lê os resultados de um banco de dados e o compara com os resultados esperados. Claramente este código precisa ser atualizado para refletir qualquer mudança feita no esquema do banco de dados.

Documentação:

Quando o esquema do banco de dados muda, a documentação que o descreve deve ser atualizada.

Como podemos ver, acoplamento é um problema sério quando tratamos de refatoração de banco de dados. Por questão de simplicidade, na continuidade deste artigo o termo “aplicação” irá se referir a

todos os sistemas externos, bancos de dados, aplicações, programas, ambientes de teste, etc., que são acoplados a um banco de dados.

Desafios Adicionais Com Sgbdrs

Acoplamento não é o único desafio que iremos nos deparar ao usar SGBDRs, apesar dele ser o mais importante. Outras questões que iremos encontrar incluem:

Questões de desempenho são difíceis de prever:

Quando estamos trabalhando com um banco de dados compartilhado, como a situação da Figura 3, podemos perceber que as características de desempenho do banco de dados são difíceis de prever porque cada aplicação acessa o banco de dados da sua forma.

Integridade de dados é difícil de garantir com bancos de dados compartilhados:

Como nenhuma aplicação simples possui controle sobre os dados, é muito difícil ter certeza de que todas as aplicações estão funcionando sob as mesmas condições.

Bancos de dados operacionais requerem estratégias de projeto diferentes daquelas aplicadas em bancos de dados para relatórios.

Os esquemas de bancos de dados operacionais refletem as necessidades operacionais das aplicações que o acessam, normalmente resultando em um esquema razoavelmente normalizado com algumas partes dele desnormalizadas por razões de desempenho. Bancos de dados de relatório, por outro lado, são tipicamente altamente desnormalizados com bastante redundância de dados para apoiar a alta demanda por relatórios.

Toda tecnologia possui seus pontos positivos e negativos, e a tecnologia de SGBDR não foge à regra. Provavelmente existem formas para mitigarmos alguns desses desafios, e encapsulamento é uma técnica importante para isso.

Encapsulamento: seu grande aliado

Encapsulamento é um recurso de projeto que trata de como uma funcionalidade é compartimentada em um sistema. Não precisamos saber como algo está implementado para usá-lo. A implicação de encapsulamento é que podemos construir qualquer coisa que desejarmos, e então podemos depois mudar a implementação e isso não afetará outros componentes no sistema (considerando que a interface para este componente não mude).

As pessoas normalmente dizem que encapsulamento é o ato de pintar a caixa de preto – estamos definindo algo que será feito, mas não estamos dizendo ao resto do mundo como ele está sendo feito. Usando o exemplo de um banco, como eles rastreiam as informações de nossas contas, em um mainframe, um mini ou um PC? Qual banco de dados eles usam? Qual o sistema operacional? Isso não importa, pois o banco encapsulou os detalhes sobre como e eles realizam os serviços nas contas. Nós apenas usamos os terminais e fazemos as operações que desejamos.

Através do acesso encapsulado a um banco de dados, talvez por algo tão simples como objetos de acesso a dados ou algo mais complexo como um framework de persistência, podemos reduzir o acoplamento do banco de dados.

A partir de agora assumimos que é possível esconder detalhes do esquema de banco de dados da maioria dos desenvolvedores na organização enquanto ao mesmo tempo damos a eles acesso ao banco de dados. Algumas pessoas, normalmente apenas DBAs responsáveis por manter o banco de dados, precisarão entender e trabalhar com o esquema de dados para manter e evoluir a estratégia de encapsulamento.

Uma vantagem do acesso encapsulado ao banco de dados é que ele permite aos programadores focar apenas no problema. Vamos assumir que estamos fazendo algo simples tal como objetos de acesso aos dados que implementem código SQL para acessar o esquema de banco de dados. Os programadores trabalharão com essas classes de acesso aos dados, não com o banco de dados. Isso permite ao DBA evoluir o esquema do banco de dados da forma que ele precisar, refatorando-o

se necessário, e tudo que ele precisa se preocupar é em manter as classes de acesso aos dados atualizadas. Isso revela uma segunda vantagem desta abordagem – ela provê uma maior liberdade para que o DBA faça seu trabalho.

A Figura 4 descreve o conceito de acesso encapsulado em banco de dados, mostrando como o cenário do melhor caso da Figura 2 e o cenário do pior caso da Figura 3 provavelmente seriam modificados. No cenário do melhor caso, os códigos fontes das regras de negócio iriam interagir com os objetos de acesso aos dados em vez de interagir com o banco de dados. A principal vantagem seria que todos os códigos relacionados a acesso a dados ficariam em um único lugar, tornando simples o processo de alteração em caso de mudanças no esquema do banco de dados ou para apoiar mudanças relacionadas a ajustes de desempenho. É interessante notar que o código com as regras de negócio que os programadores estão escrevendo estariam acoplados aos objetos de acesso aos dados. Portanto, eles precisariam ser alterados caso a interface do objeto de acesso aos dados mude.

Nós nunca ficaremos livre do acoplamento. No entanto, do ponto de vista dos programadores, é muito mais fácil mudar apenas este código – com a estratégia de encapsulamento de banco de dados eles precisariam lidar apenas com o código fonte da aplicação, e não com o código fonte do programa + código em SQL de acesso aos dados.

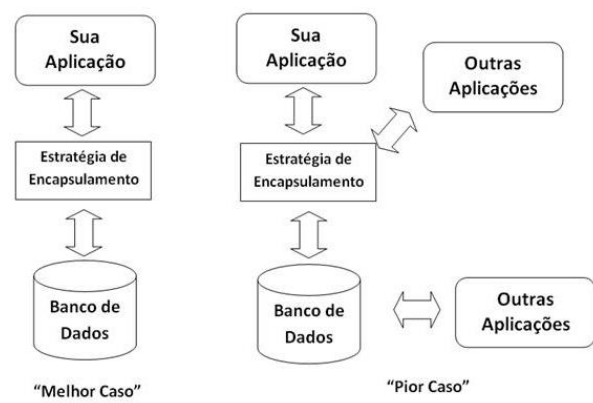


Figura 4. Os cenários revisitados.

As coisas não são assim tão ideais para o cenário do pior caso. Apesar de ser possível que todas as aplicações possam tirar vantagem da estratégia de encapsulamento, a realidade é que apenas um subconjunto estará apto a usá-lo.

Incompatibilidade de plataforma pode ser um problema – talvez os objetos de acesso aos dados são escritos em Java, mas alguns sistemas legados podem ser escritos usando tecnologias que não são tão compatíveis com Java.

Talvez você tenha optado por não refazer alguns de seus sistemas legados para simplesmente usar a estratégia de encapsulamento de dados. Talvez alguns aplicativos já tenham uma estratégia de encapsulamento em um lugar (em caso afirmativo, você pode querer considerara reutilização da estratégia existente em vez de construir sua própria). Talvez você queira usar as tecnologias que exigem acesso direto ao banco de dados.

A questão é que parte das aplicações da sua organização será capaz de tirar proveito de sua estratégia de encapsulamento e outras não. Há ainda um benefício de se fazer isso, estaremos reduzindo o acoplamento e, portanto, reduzindo seus custos de desenvolvimento e manutenção, o problema é que ele não é um benefício total realizado.

Outra vantagem do acesso encapsulado para acessar um banco de dados é disponibilizar um lugar único, além do próprio banco de dados, para implementar regras de negócio orientada a dados.

Além dos SGBDRs: você realmente tem opções

Como existem alguns problemas claros com a tecnologia de banco de dados relacional, podemos optar por usar outra tecnologia. Sim, SGBDR é o tipo de mecanismo de persistência mais comumente utilizado, mas ele não é a única opção disponível. Entre as opções, podemos citar:

Bancos De Dados Objeto/Relacional

Bancos de dados objeto/relacional (BDOR), ou mais apropriadamente sistemas gerenciadores de banco de dados objeto/relacional (SGBDORs), adicionam novas capacidades de armazenamento de objeto aos SGBDRs. BDORs adicionam novas facilidades para integrar gerenciamento de dados dos tipos tradicionais aos objetos complexos como séries temporais e dados geoespaciais e mídias binárias diversas, como áudio, vídeo, imagem e applets em Java. BDORs basicamente adicionam aos SGBDRs funcionalidades como tipos de dados, assim como a habilidade de navegar por objetos. Através da implementação de objetos em banco de dados, um SGBDOR pode executar operações analíticas complexas e manipulação de dados para buscar e transformar objetos multimídia e de outros tipos. BDORs suportam transações robustas e funcionalidades de gerenciamento de dados enquanto que ao mesmo tempo oferece uma forma limitada de flexibilidade de bancos de dados orientados a objeto.

Bancos de Dados de Objetos:

Bancos de dados de objetos (BDOs), conhecidos como bancos de dados orientados a objetos (BDOOs), quase perfeitamente adicionam funcionalidades de banco de dados/persistência aos objetos de linguagens de programação. Em outras palavras, eles trazem muito mais que armazenamento de persistência de objetos de linguagem de programação. BDOs estendem a semântica de Java para prover capacidade de programação completa de banco de dados, via novas bibliotecas específicas de classes para os BDOs disponíveis no mercado, mantendo a compatibilidade da linguagem nativa. O principal benefício desta abordagem é a unificação do desenvolvimento da aplicação e do banco de dados em um modelo “sem costura”. Como resultado, a aplicação requer menos código, usa modelagem de persistência mais natural e os códigos são mais fáceis de serem mantidos.

Além dessas opções, podemos citar ainda bancos de dados em XML, arquivos Flat (arquivos texto), bancos de dados hierárquicos e camada de prevalência. Não entraremos em detalhes sobre elas por não ser o foco principal do artigo, mas existem diversas informações disponíveis na Internet sobre estas opções.

A Tabela 2 apresenta uma comparação de vários tipos de mecanismo de persistência. A Tabela 3 apresenta sugestões para quando usar cada tipo de tecnologia.

Mecanismo	Vantagens	Desvantagens	Aplicação Potencial
Arquivos Flat	Abordagem simples para persistência Solução boa para sistemas pequenos A maioria das linguagens possui suporte para arquivos textuais Não requer licença de uso	Difícil acesso ad-hoc	Aplicações simples, particularmente aquelas com um paradigma “leia todas as informações, manipule-as e salve-as em disco” Persistência de informação de configuração Compartilhamento de informação com outro sistema Auditar arquivo de log e sistema de relatório
Bancos de dados hierárquicos	Suporta aplicações orientadas a transações	Não é tão popular	Aplicações orientadas a transação Fonte comum de dados legados
Bancos de dados de objeto	Abordagem “pura” para persistir objetos Excelente opção para um banco de dados de aplicação específica quando a tecnologia OO é usada	Não é bem aceito no mercado e os padrões definidos, como Object Query Language(OQL), ainda estão evoluindo	Estruturas de dados altamente inter-relacionadas e complexas Transações complexas Aplicações simples, família de aplicações ou linha de produto

	Abordagem uniforme para armazenamento da aplicação e dados Facilidade de refatoração, pois tudo é objeto		
Bancos de dados objeto/relacional	Tecnologia em crescimento	Não é bem aceito no mercado, padrões emergentes, como SQL3, não são largamente adotados e possui base de experiência ainda pequena	Estruturas de dados altamente inter-relacionadas Transações complexas Aplicações simples, família de aplicações ou linha de produto
Camada de Prevalência	Persistência transparente de objetos Desempenho Simplicidade	Tecnologia emergente	Estrutura de objetos complexa Aplicações simples, família de aplicações ou linha de produto
Banco de dados relacional	Tecnologia madura Mecanismo de persistência dominante no mercado Muitos produtos disponíveis Padrões, como SQL e JDBC, bem definidos e aceitos Base de experiência de desenvolvedores extensa	Mapeamento de objeto para banco de dados relacional pode ser uma habilidade difícil de ser aprendida	Aplicações orientadas a transação Complexidade de dados de simples para intermediária Aplicação com grande carga de dados Banco de dados compartilhado e operacional Banco de dados de relatório
Bancos de dados em XML	Suporte nativo para persistir estruturas de dados em XML Para aplicações que usam XML, ele remove a necessidade de navegação entre as estruturas do XML e do banco de dados	Tecnologia emergente com padrões, como o XML equivalente de SQL, não tão adotado e não funciona bem para sistemas orientados a transação	Ambiente ideal para aplicações que usam XML, tal como portais ou facilidades para relatórios online

Tabela 2. Comparando Mecanismos de Persistência.

A tecnologia de SGBDRs não é perfeita, como nenhuma tecnologia é, mas ela é uma das mais utilizadas em nossa área, então precisamos aprender como ela funciona efetivamente. A razão pela qual discutimos sobre os pontos negativos desta tecnologia é que eles nos permitem conhecer as limitações para usar tal tecnologia em nosso dia-a-dia.

Em geral, alguns autores sempre focam nos benefícios do uso de SGBDRs, e claramente existem muitos, mas ignoram os pontos negativos. Outros autores focam em questões acadêmicas, deixando um pouco de lado questões práticas, que é como de fato aprendemos a usar uma tecnologia.

Acoplamento é uma questão séria para todos profissionais de TI, incluindo desenvolvedores e DBAs. Acesso encapsulado a um banco de dados pode ajudar a minimizar os problemas de acoplamento, mas ele é apenas uma solução parcial.

É importante também reconhecer que bancos de dados relacionais são apenas uma das várias escolhas que temos disponíveis para persistência dos dados.

Abordagens não-relacionais são soluções viáveis para algumas situações e devem ser levadas em consideração. De qualquer forma, bancos de dados relacionais serão sempre soluções para trabalharmos com persistência de dados.

Linguagens De Definição E Manipulação De Dados

Usando a linguagem de definição de dados (DDL)

O **SQL Server** (o enfoque será sobre a versão 2000) é um **SGBD** - Sistema Gerenciador de Banco de dados - da Microsoft (originalmente o projeto do SQL Server foi desenvolvimento pela Sybase) que pode ser instalado no Windows NT/2000 e Win9x e que possui as seguintes características :

- **É fácil de usar (se comparado com outros SGBD)**
- **Oferece escalabilidade , ou seja, você pode começar desenvolvendo para um desktop e migrar para sistemas de multiprocessamento sem traumas.(bem com poucos traumas...)**
- **Implementa o data warehouse , através do Analysis Services.(antes somente disponível no Oracle e demais SGBD)**
- **É relativamente barato (se comparado com outros SGBD)**

Geralmente dizemos que o SQL Server é um SGBD **cliente/Servidor** pois comporta diferentes tipos de plataformas e possui funcionalidades divididas entre clientes e servidores , onde o cliente fornece uma ou mais interfaces que serão usadas para requerer uma solicitação ao servidor(SGBD) ; este por sua vez , processa a solicitação e devolve o resultado ao cliente.

O SQL Server possui uma linguagem relacional chamada de **Transact-SQL** que é um dialeto da linguagem **SQL - Structured Query Language**. A principal característica da linguagem SQL é ter sido projetada para trabalhar com conjuntos de registros de dados , enquanto que as linguagens tradicionais (C++, VB , Delphi,...) podem tratar apenas um registro por vez. Além disto a SQL não é procedural , ou seja , a SQL não precisa descrever em detalhes como realizar uma tarefa , ela apenas descreve o que o usuário final deseja.

A SQL se divide em dois subconjuntos de linguagem :

1. **A linguagem de Definição de Dados - DDL - que usa instruções para descrever o esquema das tabelas do banco de dados**
2. **A linguagem de Manipulação de Dados - DML - que usa instruções para manipular os dados.**

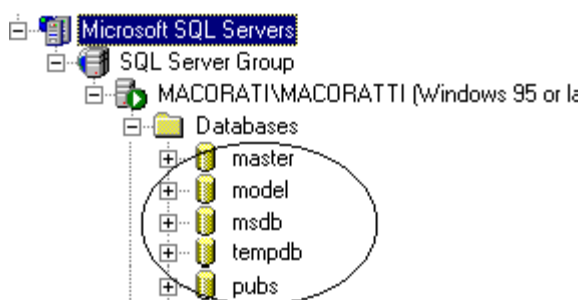
Usando a DDL - Criando objetos de banco de dados

Na organização do SQL Server podemos ter objetos físicos e lógicos :

- i.Os objetos físicos relacionam-se com a organização dos dados no disco
- ii.Os objetos lógicos são constituídos por : banco de dados , tabelas , visões , colunas , linhas , etc...

O SQL Server permite a criação de banco de dados do usuário (criados por um usuário autorizado) e banco de dados do sistema (criados durante a instalação) os quais são :

- **Master**
- **tempdb**
- **model**
- **msdb**
- **distribution**



Para criar um banco de dados podemos usar o - SQL Server Enterprise Manager - ou usar uma instrução **Transact-SQL** , a instrução **CREATE DATABASE** cuja sintaxe é :

```
CREATE DATABASE nome_do_bd  
[ ON PRIMARY ] arquivo1 , arquivo2, ...  
[ LOG ON arquivo3, arquivo4,... ]  
[ FOR RESTORE ]  
[ COLLATE nome_collate ]
```

I.**nome_do_bd** - é o nome do banco de dados (podemos usar até 128 caracteres)

II.**ON PRIMARY** - especifica o primeiro arquivo que contém tabelas de sistema e informações internas

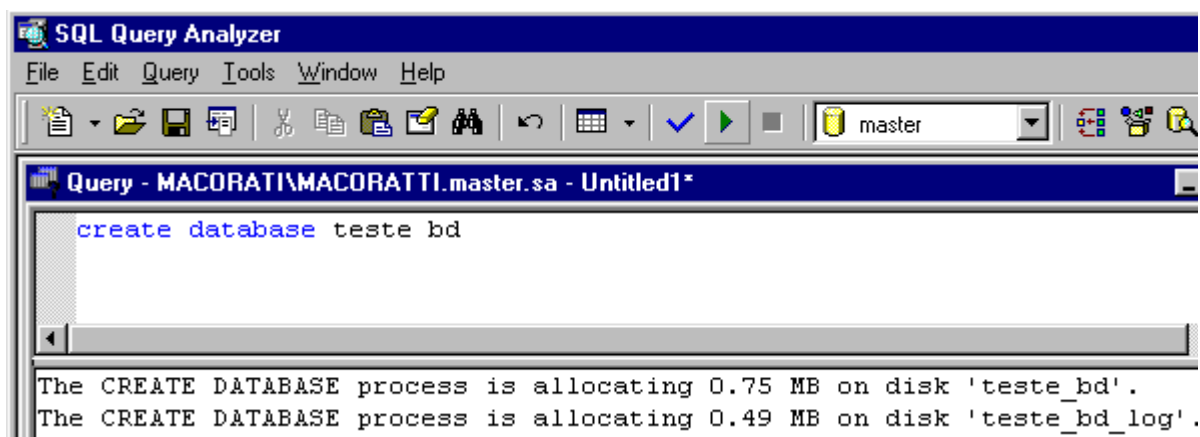
i.arquivo1 , arquivo2 - representam as especificações dos arquivos (nome, tamanho, etc...)

III.**LOG ON** - é usada para definir um ou mais arquivos como destino do log de transações do banco de dados.

IV.**FOR RESTORE** - usada apenas por questões de compatibilidade com as versões anteriores

V.**COLLATE** - define um conjunto de regras padrão para o banco de dados.

Vamos criar um banco de dados simples que não use nenhuma especificação. Abra o - SQL Query Analyser - e digite a instrução : **CREATE DATABASE teste_bd** e clique no botão para executar a query. Você verá:



Criamos o banco de dados **teste_bd**

- o nome lógico é **teste_bd**

- o nome lógico do Log de transações é : **teste_bd_log**

Se quisermos especificar detalhes na criação de um banco de dados podemos fazer :

```
CREATE DATABASE teste_bd  
ON (NAME=teste_dat,  
FILENAME = 'C:\SQLSV\teste_bd.mdf',  
SIZE = 5,  
MAXSIZE= 20,
```

FILEGROWTH=5)

LOG ON

(NAME= teste_log,

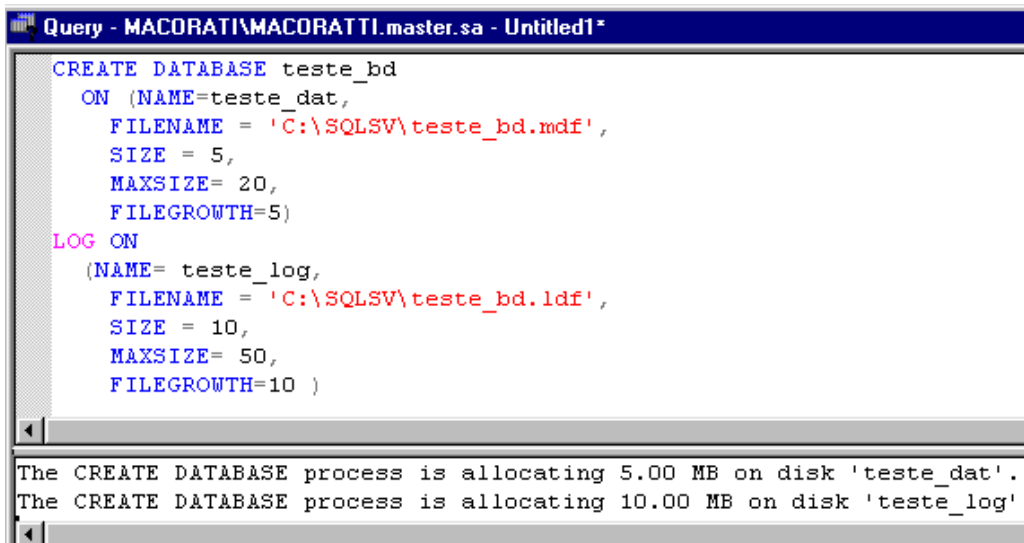
FILENAME = 'C:\SQLSV\teste_bd.ldf',

SIZE = 10,

MAXSIZE= 50,

FILEGROWTH=10)

Executando o código no - **Query Analyser** - obtemos o seguinte resultado:



```

Query - MACORATI\MACORATTI.master.sa - Untitled1*
CREATE DATABASE teste_bd
ON (NAME=teste_dat,
    FILENAME = 'C:\SQLSV\teste_bd.mdf',
    SIZE = 5,
    MAXSIZE= 20,
    FILEGROWTH=5)
LOG ON
(NAME= teste_log,
    FILENAME = 'C:\SQLSV\teste_bd.ldf',
    SIZE = 10,
    MAXSIZE= 50,
    FILEGROWTH=10 )

The CREATE DATABASE process is allocating 5.00 MB on disk 'teste_dat'.
The CREATE DATABASE process is allocating 10.00 MB on disk 'teste_log'.

```

No exemplo acima estamos criando um banco de dados chamado - **teste_bd** . Como omitimos a opção **PRIMARY** assumimos que o primeiro arquivo é o primário. Este arquivo tem o nome lógico de **teste_dat** e esta armazenado no arquivo **teste_bd.mdf** com tamanho original de 5 MB.

Temos também um arquivo de log de transações com o nome lógico de **teste_log** e nome físico de **teste_bd.ldf**. (para usar o banco de dados basta abrí-lo usando o comando USE teste_bd da T-SQL)

Nota:

É possível criar um banco de dados no SQL Server usando código Visual Basic ? A resposta é NÃO.

Para criar um banco de dados no VB usamos a **ADOX**. E não podemos usar ADOX para criar um banco de dados SQL Server. Mas há uma alternativa (sempre há ... 😊).

Podemos usar uma conexão ADO para fazer isto, veja o código abaixo:

```

Dim oConn As ADODB.Connection
Dim sDatabaseName As String

```

```

sDatabaseName = "teste1"

```

```

Set oConn = New ADODB.Connection
oConn.Open "Provider=SQLOLEDB;Data Source=(local);User ID=sa;Password=;"
oConn.Execute "CREATE DATABASE " & sDatabaseName

```

Criando Tabelas

A instrução **Create Table** cria uma nova tabela . A forma básica da instrução é a seguinte:

```
CREATE TABLE nome_tabela  
(nome_col1 tipo1 [NOT NULL | NULL]  
nome_col2 tipo2 [NOT NULL | NULL]
```

- **nome_tabela** - é o nome da tabela que deverá ser criada (o número máximo de tabelas em um banco de dados é limitado pelo número de objetos no banco de dados. Podemos de mais de 2 bilhões de objetos em um bd)

- **nome_col1** , **nome_col2** - são os nomes das colunas da tabela.(o número máximo de colunas é 1024)

- **tipo1** , **tipo2** - são os tipos de dados para cada coluna.

A existência de valores Nulos em uma coluna é definida pelas cláusulas : **NOT NULL** ou **NULL**.

Se **NOT NULL** estiver definido , a atribuição de valores nulos para a coluna não é permitida ; se houver uma tentativa de inserir um valor nulo ocorrerá uma mensagem de erro.

- O privilégio para criar uma tabela após a criação do banco de dados é concedido de forma implícita ao Administrador do sistema e ao proprietário do banco de dados.(GRANT CREATE TABLE concede privilégios para criar tabelas a outros usuários)

- Além das tabelas de base existem alguns tipos especiais de tabelas : temporárias (armazenadas no banco de dados **tempdb**) e as visualizações.

Vejamos como criar uma tabela usando **Create Table**:

```
CREATE TABLE alunos (cod_aluno INTEGER NOT NULL,  
nome_aluno CHAR(30) NOT NULL,  
endereco_aluno CHAR(50) NOT NULL,  
telefone_aluno CHAR(20) NULL,  
nascimento_aluno DATETIME NULL)
```

O código acima cria a tabela **alunos** com a seguinte estrutura :

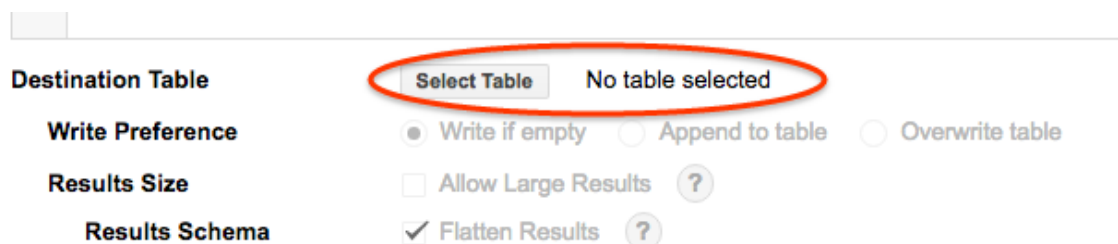
Nome da coluna	Tamanho da Coluna	Permite Nulos
cod_aluno	-	Não
nome_aluno	30	Não
endereco_aluno	50	Não
telefone_aluno	20	Sim
nascimento_aluno	-	Sim

Linguagem De Manipulação De Dados

A Linguagem de manipulação de dados (DML, na sigla em inglês) do BigQuery permite atualizar, inserir e excluir dados das suas tabelas do BigQuery.

Você pode executar declarações DML como se fossem uma declaração SELECT, com as seguintes condições:

- É necessário usar o SQL padrão. Para ativá-lo, consulte Ativação do SQL padrão.
- Não é possível especificar uma tabela de destino. Por exemplo, na IU da Web, é necessário definir **Tabela de destino** para **Nenhuma tabela selecionada**.



Cotas

O seguinte limite se aplica à Linguagem de manipulação de dados (DML, na sigla em inglês):

- **Máximo de instruções UPDATE/DELETE por dia por tabela:** 96.
- **Máximo de instruções UPDATE/DELETE por dia por projeto:** 1.000.
- **Máximo de instruções INSERT por dia por tabela:** 1.000.
- **Máximo de instruções INSERT por dia por projeto:** 1.000.

O processamento das instruções DML é consideravelmente mais caro em comparação às instruções SELECT.

Para informações sobre cotas do BigQuery, consulte Política de cotas.

Preços

O BigQuery cobra por consultas DML com base no número de bytes processados pela consulta. O número de bytes processados é calculado da maneira a seguir.

Ação	Notas
INSERT	Bytes processados = soma de bytes processados referentes a todos os campos mencionados nas tabelas verificadas pela consulta.
ATUALIZAR	<p>Bytes processados = soma de bytes nos campos mencionados nas tabelas verificadas + a soma de bytes de todos os campos na tabela atualizada no momento em que UPDATE se inicia.</p> <p>Exemplo 1:</p> <p>"Table1" tem dois campos, "col1" do tipo inteiro e "col2" do tipo string.</p> <p>UPDATE table1 SET col1 = 1 WHERE col1 = 2;</p> <p>Bytes processados no exemplo 1: soma do número de bytes em "col1" + soma do número de bytes em "col2".</p> <p>Exemplo 2:</p>

	<p>"Table1" tem dois campos, "col1" do tipo inteiro e "col2" do tipo string, e "table2" tem um campo, "field1", do tipo inteiro.</p> <p>UPDATE table1 SET col1 = 1 WHERE col1 in (SELECT field1 from table2)</p> <p>Bytes processados no exemplo 2: soma do número de bytes em "table1.col1" antes de UPDATE + soma do número de bytes em "table1.col2" antes de UPDATE + soma do número de bytes em "table2.field1"</p>
DELETE	<p>Bytes processados = soma de bytes dos campos mencionados nas tabelas verificadas + a soma de bytes de todos os campos na tabela modificada no momento em que DELETE se inicia.</p>

Para outras informações sobre preços do BigQuery, consulte Preços.

Problemas Conhecidos

- Uma instrução DML inicia uma transação implícita. Isso significa que a confirmação das alterações feitas por ela é feita automaticamente no final de cada instrução DML bem-sucedida. Transações de múltiplas instruções não são aceitas.
- Somente as seguintes combinações de instruções DML têm permissão para serem executadas simultaneamente em uma tabela:
 - UPDATE e INSERT
 - DELETE e INSERT
 - INSERT e INSERT

Caso contrário, uma das instruções DML será cancelada. Por exemplo, se duas instruções UPDATE forem executadas simultaneamente em relação a uma tabela, só uma delas será bem-sucedida.

- Tabelas que receberam gravações recentemente por meio do BigQuery Streaming (tabledata.insertall) não podem ser modificadas por instruções UPDATE ou DELETE. Para verificar se a tabela tem um buffer de streaming, procure uma seção chamada "streamingBuffer" em "tables.get response". Se não encontrar, a tabela pode ser modificada por instruções UPDATE ou DELETE.
- As instruções DML que modificam tabelas particionadas ainda não são aceitas.

Capítulo 6. Manipulação De Dados

O capítulo anterior mostrou como criar tabelas e outras estruturas para armazenar dados. Agora está na hora de preencher as tabelas com dados. Este capítulo mostra como inserir, atualizar e excluir dados em tabelas. Também são apresentadas maneiras de efetuar mudanças automáticas nos dados quando ocorrem certos eventos: gatilhos (triggers) e regras de reescrita (rewrite rules). Para completar, o próximo capítulo explica como fazer consultas para extrair dados do banco de dados.

Inserção De Dados

A tabela recém-criada não contém dados. A primeira ação a ser realizada para o banco de dados ter utilidade é inserir dados. Conceitualmente, os dados são inseridos uma linha de cada vez. É claro que é possível inserir mais de uma linha, mas não existe maneira de inserir menos de uma linha por vez. Mesmo que se conheça apenas o valor de algumas colunas, deve ser criada uma linha completa.

Para criar uma linha é utilizado o comando INSERT. Este comando requer o nome da tabela, e um valor para cada coluna da tabela. Por exemplo, considere a tabela produtos do Capítulo 5:

CREATE TABLE produtos (

cod_prod integer,

nome text,

preco numeric

);

Um exemplo de comando para inserir uma linha é:

```
INSERT INTO produtos VALUES (1, 'Queijo', 9.99);
```

Os valores dos dados são colocados na mesma ordem que as colunas se encontram na tabela, separados por vírgula. Geralmente os valores dos dados são literais (constantes), mas também são permitidas expressões escalares.

A sintaxe mostrada acima tem como desvantagem ser necessário conhecer a ordem das colunas da tabela. Para evitar isto, as colunas podem ser relacionadas explicitamente. Por exemplo, os dois comandos mostrados abaixo possuem o mesmo efeito do comando mostrado acima:

```
INSERT INTO produtos (cod_prod, nome, preco) VALUES (1, 'Queijo', 9.99);
```

```
INSERT INTO produtos (nome, preco, cod_prod) VALUES ('Queijo', 9.99, 1);
```

Muitos usuários consideram boa prática escrever sempre os nomes das colunas.

Se não forem conhecidos os valores de todas as colunas, as colunas com valor desconhecido podem ser omitidas. Neste caso, estas colunas são preenchidas com seu respectivo valor padrão. Por exemplo:

```
INSERT INTO produtos (cod_prod, nome) VALUES (1, 'Queijo');
```

```
INSERT INTO produtos VALUES (1, 'Queijo');
```

A segunda forma é uma extensão do PostgreSQL, que preenche as colunas a partir da esquerda com quantos valores forem fornecidos, e as demais com o valor padrão.

Para ficar mais claro, pode ser requisitado explicitamente o valor padrão da coluna individualmente, ou para toda a linha:

```
INSERT INTO produtos (cod_prod, nome, preco) VALUES (1, 'Queijo', DEFAULT);
```

```
INSERT INTO produtos DEFAULT VALUES;
```

SCRUM

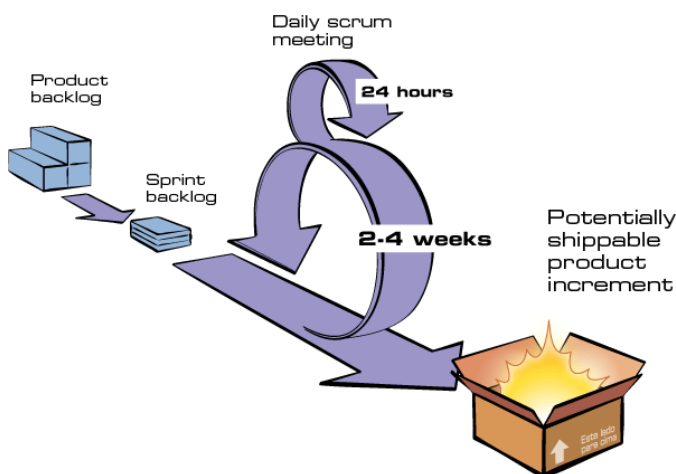
Scrum é uma **metodologia ágil** para gestão e planejamento de projetos de software.

No Scrum, os projetos são divididos em ciclos (tipicamente mensais) chamados de **Sprints**. O **Sprint** representa um Time Box dentro do qual um conjunto de atividades deve ser executado. **Metodologias ágeis** de desenvolvimento de software são iterativas, ou seja, o trabalho é dividido em iterações, que são chamadas de Sprints no caso do Scrum.

As funcionalidades a serem implementadas em um projeto são mantidas em uma lista que é conhecida como **Product Backlog**. No início de cada Sprint, faz-se um **Sprint Planning Meeting**, ou seja, uma reunião de planejamento na qual o **Product Owner** prioriza os itens do **Product Backlog** e a equipe seleciona as atividades que ela será capaz de implementar durante o Sprint que se inicia. As tarefas alocadas em um Sprint são transferidas do **Product Backlog** para o **Sprint Backlog**.

A cada dia de uma Sprint, a equipe faz uma breve reunião (normalmente de manhã), chamada **Daily Scrum**. O objetivo é disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia.

Ao final de um Sprint, a equipe apresenta as funcionalidades implementadas em uma **Sprint Review Meeting**. Finalmente, faz-se uma **Sprint Retrospective** e a equipe parte para o planejamento do próximo Sprint. Assim reinicia-se o ciclo. Veja a ilustração abaixo:



O que é Scrum:

Scrum é uma **metodologia** usada para a **gestão dinâmica de projetos**, sendo muitas vezes aplicada para o **desenvolvimento ágil de um software**.

O scrum é uma ferramenta que permite controlar de forma eficaz e eficiente o trabalho, potencializando as equipes que trabalham em prol de um objetivo em comum.

Esta metodologia é essencial para muitas empresas atualmente, porque não apenas facilita a definição de objetivos, como também ajuda a cumprir os prazos estabelecidos.

No scrum se trabalha com o chamado product backlog, um registro que contém as áreas do produto que devem ser desenvolvidas. Do product backlog é criado o release backlog, que é a junção dos requisitos do product backlog que vão ser trabalhados, de acordo com a prioridade de cada um. O release backlog é um ponto para a criação do sprint backlog, que representa o espaço de tempo em que uma tarefa (chamada de user story) vai ser concluída.

O tamanho de cada sprint é adequado à empresa em questão e aos seus projetos. A sprint pode demorar entre uma a quatro semanas. O processo de scrum costuma ser controlado em um quadro, onde é possível ver as tarefas que estão em desenvolvimento, as que foram trabalhadas, mas que ainda precisam ser verificadas ou testadas, e as que são consideradas concluídas.

Alguns dos elementos que fazem parte do processo do Scrum são:

Product owner: é o dono do produto ou projeto que vai ser trabalhado, sendo responsável pela direção a seguir, definindo quais requisitos vão fazer parte do product backlog e quais devem ser abordados pela equipe. Representa os usuários ou clientes do produto em questão;

Scrum Master: é o elemento que faz a ligação entre o product owner e a equipe. Tem a responsabilidade de organizar reuniões, fazer o acompanhamento do trabalho e se certificar que cada integrante da equipe tem as ferramentas necessárias para cumprir a sua função da melhor maneira possível.

Team (equipe): É a equipe que trabalha para o desenvolvimento do projeto ou produto.

Outro conceito relevante nesta área é o **daily scrum**, ou scrum diário, que consiste em uma reunião organizada pelo Scrum Master. Todos os elementos estão em pé, para que a reunião seja de curta duração (máximo 15 minutos). Esta reunião é uma forma de comprovar que cada elemento está cumprindo o seu papel.

Outra reunião importante no âmbito do scrum é a de planejamento da próxima sprint, onde é definido quanto tempo vai durar cada tarefa. O standard para a medição do tempo de cada tarefa pode ser atribuição de pontos ou tamanhos de camiseta (XL, L, M, S, XS), sendo que uma tarefa que demora mais tem mais pontos. Desta forma, é possível somar os pontos no fim da sprint e averiguar a velocidade de trabalho da equipe.

A monitorização do progresso de cada sprint é feita através da **burndown chart** (tabela burndown), uma das características que torna o scrum tão popular.

Consiste em uma tabela que permite controlar se um projeto está se desenvolvendo da forma programada. Ela apresenta uma medição diária da quantidade de trabalho que ainda não foi feito em cada sprint ou release. Esta tabela também permite fazer uma estimativa do tempo em que a sprint vai ser concluída. Assim, é possível saber se o projeto está progredindo de acordo com o tempo estimado ou se vai sofrer algum atraso. Essa informação pode ser usada pela equipe para fazer alguns ajustes no seu trabalho, impedindo que o atraso se verifique realmente.

A origem do termo scrum vem do esporte rúgbi, onde scrum define a aglomeração dos jogadores, muitas vezes vista como "formação ordenada". No scrum, 8 jogadores de cada time estão frente a frente e têm que fazer um esforço para recuperar a bola que se encontra no meio do "aglomerado".

A metodologia Agile, como o próprio nome já diz, é utilizada para tornar os processos empresariais mais ágeis, sobretudo o desenvolvimento de sistemas. Dentro das metodologias ágeis, o framework Scrum é um dos mais difundidos e utilizados. Juntos permitem controlar de forma eficiente as atividades realizadas, incentivando as equipes a trabalharem com foco em um objetivo comum.

Além de otimizar a definição de metas, o Scrum assegura a geração de valor em um projeto, uma forma de trabalho inovadora que tem sido adotada por grandes empresas.

O que é metodologia Agile

A metodologia Agile, ou ágil em português, se consolidou nos últimos anos como uma alternativa para atender às demandas de clientes e projetos de forma dinâmica, flexível e com grande aumento de produtividade.

No desenvolvimento ágil, é utilizada uma abordagem de planejamento iterativa. Enquanto no método tradicional todas as etapas do projeto são documentadas detalhadamente, desde o início até o fim do projeto, esse processo no método ágil é realizado em etapas curtas, chamadas iterações.

Fundamentos da metodologia Agile

A metodologia Agile, nos moldes como é conhecida atualmente, foi concebida no início de 2001. Um grupo de 17 conceituados desenvolvedores de software se reuniu para aprimorar conceitos e metodologias ágeis existentes e formular o "Manifesto para o Desenvolvimento Ágil de Software". Assinado pelos 17 desenvolvedores, ele reúne quatro valores e 12 princípios.

Os valores determinam o que deve ser priorizado:

Os indivíduos e as interações entre eles mais que os processos e as ferramentas;

O software funcionando mais do que uma documentação completa e abrangente;

A colaboração com e dos clientes mais do que as negociações de contratos; e

Respostas às mudanças mais do que seguir o plano inicial.

Já os princípios são:

A maior prioridade é satisfazer o cliente com a entrega adiantada e contínua de software de valor;

Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças para que o cliente possa tirar vantagens competitivas;

Entregar software funcionando com frequência, preferencialmente em semanas;

Cooperação diária entre pessoas que entendem do "negócio" e desenvolvedores;

Projetos surgem por meio de indivíduos motivados, entre os quais existe relação de confiança.

A maneira mais eficaz e eficiente de transmitir informações é por conversas frente a frente;

Softwares funcionais são a principal medida de progresso do projeto;

Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter, indefinidamente, passos constantes;

Contínua atenção à excelência técnica e bom design aumenta a agilidade;

Simplicidade é essencial. Cultivar a arte de maximizar a quantidade de trabalho que não precisou ser feito;

As melhores arquiteturas, requisitos e projetos emergem de equipes auto-organizadas;

Em intervalos regulares, o time reflete sobre como se tornar mais efetivo e, então, se ajustam e otimizam seu comportamento de acordo.

O funcionamento do framework Scrum

Entre as diversas metodologias ágeis usadas por desenvolvedores, o framework Scrum é uma das mais difundidas, especialmente pelo formato dinâmico como as etapas dos projetos são desenvolvidas.

Em um projeto realizado utilizando o Scrum, a execução acontece em iterações, as chamadas sprints. Elas são ciclos de desenvolvimento que começam em uma reunião de planejamento (Sprint Planning) e terminam com outros dois eventos: a revisão da sprint (Sprint Review), em que o produto é demonstrado; e a retrospectiva da sprint (Sprint Retrospective), em que o time afia seus processos ao refletir sobre os aprendizados que tiveram naquele ciclo. O desenvolvimento é acompanhado por reuniões diárias em pé, as chamadas Daily Scrum.

Uma sprint pode durar qualquer unidade de tempo, mas a recomendação é que as mais curtas levem uma semana e as mais longas, um mês. Quando um time define um padrão de duração da Sprint, ele costuma ser usado em todas as iterações do projeto. As sprints são sucessivas e, assim, a seguinte somente é iniciada ao fim da anterior.

No Scrum, três papéis são definidos para a divisão dos membros envolvidos num projeto:

Product Owner: atua como o "dono" do projeto. Será o responsável por definir prioridades a serem desenvolvidas em cada sprint e fará a intermediação entre a área de negócios e a equipe de scrum.

Scrum Master: atua como um líder-servo, blindando os demais membros e assegurando que a equipe siga a metodologia do Scrum sem interrupções externas. É também o responsável por

remover obstáculos que possam prejudicar o desenvolvimento realizado pela equipe e ajudá-la a cumprir suas tarefas com a melhor performance possível.

Scrum Team: a equipe de desenvolvimento. Todos devem se comprometer em realizar as entregas estabelecidas dentro de uma Sprint. Para tal, é necessário que haja maturidade suficiente de cada membro para a execução de sua função e, quando preciso, solicite auxílio. Para a eficiência do Scrum, é recomendável que essas equipes sejam compostas por membros multidisciplinares e não envolvam muitos participantes.

Sprint Planning

O planejamento da sprint é realizado nessa reunião, em que será definido como o trabalho da equipe será feito dentro do período estabelecido. Nesse momento, devem ser priorizadas as atividades do product backlog — descrições de todas as funcionalidades desejadas para o produto — que passarão a integrar o sprint backlog.

É importante ressaltar que o backlog é composto pelo repositório de projetos e ações da empresa. Em cada sprint, seleciona-se, de acordo com a prioridade e dificuldade do projeto, os projetos que sairão do backlog e comporão a sprint.

Daily Scrum

Diariamente, a equipe de desenvolvimento deve se reunir para discutir aquilo que tem sido desenvolvido dentro da sprint. Essa reunião deve ser rápida e, preferencialmente, não durar mais que 15 minutos. Nela, serão respondidas três perguntas por cada membro da equipe:

O que foi feito ontem para ajudar a equipe de desenvolvimento a atender a meta da sprint?

O que será feito hoje para ajudar a equipe de desenvolvimento a atender a meta da sprint?

Há algum obstáculo que impeça o atendimento da meta da sprint?

Se houver uma resposta afirmativa para a última pergunta, o Scrum Master deverá buscar rapidamente uma solução para que o desenvolvimento da sprint siga conforme o planejado. Por isso é importante que ele tenha autonomia na organização para corrigir problemas no processo.

Sprint Meeting Review

Ao final da sprint, uma reunião de revisão é realizada para a discussão daquilo que foi desenvolvido naquele ciclo. Caberá ao Product Owner analisar se cada tarefa foi concluída conforme o esperado ou se alguma delas deverá retornar ao Product Backlog para a inclusão em uma nova sprint.

Sprint Retrospective

A reunião de retrospectiva da sprint é realizada após a reunião de revisão e anteriormente à reunião de planejamento, para que seja discutido um plano de melhorias. O objetivo dessa reunião é promover a colaboração entre os membros da equipe de desenvolvimento para corrigir possíveis desvios de rota e aprimorar aquilo que já está nos trilhos.

Por que as empresas utilizam o Scrum?

Em qualquer gerenciamento de projeto, os pilares buscados são a diminuição de custos, entregas mais rápidas e com maior qualidade. Em suma, as empresas que recorrem ao Scrum miram esses três objetivos numa única metodologia.

De forma geral, os projetos são desenvolvidos para a criação ou melhoria de um produto, pensando em necessidades de mercado que podem se transformar rapidamente, de acordo com o segmento de cada cliente. Com as entregas parciais a cada sprint, o Scrum permite que o projeto seja construído aos poucos e novas funcionalidades possam ser integradas. Dessa forma, a metodologia permite que os produtos sejam lançados, testados e validados no mercado rapidamente, reduzindo o “time-to-market”.

Uma vez que o Product Owner priorizará atividades mais importantes para a realização do projeto, haverá a segurança de que os requisitos de maior valor para o negócio terão sua entrega privilegiada pelo time de desenvolvimento. Ou seja, aquilo que é primordial para o cliente receberá atenção especial, enquanto funcionalidades secundárias tendem a levar mais tempo para entrar em uma sprint. Por consequência, o Scrum agrega maior valor ao negócio e em menor tempo, o que diminui os custos de operação e potencializa o ROI desses projetos.

Da mesma forma em que são estabelecidas prioridades para o desenvolvimento no projeto, o Scrum abre espaço para atender à necessidade de mudanças no decorrer desse processo, na medida em que o desenvolvimento é realizado em curtas etapas, com entregas parciais.

Enquanto as metodologias tradicionais não proporcionavam a liberdade para que alterações fossem implementadas durante o projeto — era necessário aguardar a entrega final para testar suas aplicações —, os desenvolvedores estão mais capacitados a absorver a necessidade de mudanças com a adoção de metodologias ágeis e podem modificar o escopo do projeto.

Nesse sentido, há um claro aumento de competitividade das empresas que recorrem ao Scrum. Com as entregas parciais daquilo que é desenvolvido e os testes realizados ao fim de cada sprint, há a possibilidade de o projeto ser modificado para atender a novas demandas do cliente e as prioridades serem alteradas no backlog já no próximo ciclo de produção.

Metodologia Scrum e Agile têm se consolidado como alternativas para o desenvolvimento de sistemas com rapidez e eficiência, permitindo não apenas a redução de tempo, como também de gastos. Todavia, o gestor de projetos deverá conhecer diferentes técnicas, a fim de definir aquela que será a mais adequada para as necessidades de sua empresa. Nesse sentido, é possível aplicar Scrum e outras metodologias ágeis em soluções híbridas.

Desenvolvimento ágil com Scrum: uma visão geral

Considerando o cotidiano da maioria das organizações na atualidade, é inegável a importância desempenhada por sistemas computacionais dos mais variados tipos na condução de operações rotineiras. Esse suporte não apenas contribui para uma maior eficácia na execução de atividades do dia-a-dia, como também foi um fator determinante para um incremento substancial na produtividade de diversos processos de negócio.

A evolução da área de sistemas foi acompanhada pelo surgimento de diversas metodologias, com estas últimas normalmente englobando um conjunto de diretrizes e conceitos criados com o intuito de nortear o processo de construção de um software. Como de praxe, não há uma fórmula mágica para se chegar ao resultado final, ou seja, uma aplicação que atenda às expectativas dos usuários e seja capaz de funcionar dentro de uma série de parâmetros considerados como aceitáveis. Partindo de um conjunto de técnicas e diretrizes com eficácia já comprovada, os profissionais envolvidos empreendem esforços no sentido de adaptar um modelo para a realidade na qual se encontram inseridos.

É importante frisar sempre que desenvolver softwares não é uma tarefa das mais simples. Uma ampla gama de variáveis influencia no modo como uma aplicação será construída, somando-se ainda a isto influências como a pressão pela entrega do produto em um prazo muito curto, mudanças motivadas por alterações na legislação vigente, dificuldades dos usuários que solicitam um sistema em descrever de forma clara e concisa aquilo que realmente necessitam (com prováveis pedidos de modificações ao longo do projeto), dentre outros aspectos.

A noção de mudança também representa um elemento central na elaboração de aplicações. Por mais que todo um esforço procurando contemplar um número extenso de situações seja levado a cabo, é praticamente impossível construir um software que em determinado momento não precise passar por alterações. Aliás, é bastante comum que exista a necessidade de modificações durante a construção da aplicação, comprometendo assim uma parcela do tempo e de recursos que já haviam sido alocados para a implementação daquela solução.

Em um cenário de transformações constantes, a escolha pelo paradigma de desenvolvimento mais adequado a um determinado contexto é, sem sombra de dúvidas, um fator crucial para o sucesso do projeto. As primeiras técnicas formais voltadas à construção de sistemas possuíam um enfoque direcionado à elaboração de soluções com uma estrutura mais rígida e, portanto, eram menos

suscetíveis a mudanças. A necessidade de uma rápida adaptação diante de situações imprevistas é um dos aspectos que caracteriza o mundo corporativo atual; procurando atender a este tipo de demanda, surgiriam práticas mais flexíveis para a criação de aplicações, com diversas destas sendo conhecidas como “metodologias ágeis”.

O objetivo deste artigo é abordar, em linhas gerais, como métodos ágeis podem ser empregados em atividades relacionadas com o desenvolvimento de sistemas. Isto será feito, basicamente, através de um estudo mais detalhado de um conjunto de práticas com uma grande aceitação na área de software atualmente: trata-se da metodologia Scrum.

Metodologias de Desenvolvimento de Software: uma Visão Geral

O modelo em cascata (em inglês “waterfall”) foi, ainda na década de 1970, um dos primeiros padrões a fornecer diretrizes para processos voltados ao desenvolvimento de software. Esta metodologia é caracterizada por fases que ocorrem dentro de uma sequência rígida, com o início das atividades de uma etapa acontecendo imediatamente após o término daquela que a precedeu. A implementação de um projeto que segue este modelo é geralmente dividida nas seguintes fases: análise de requisitos, projeto da aplicação, implementação, testes de validação, implantação e manutenção.

De certa forma, esta abordagem é bastante semelhante à da linha de montagem clássica do mundo fabril. Considerando o cenário atual, em que muitas organizações se veem às voltas de transformações profundas e muitas vezes repentinas, a maneira linear que este modelo impõe à atuação dentro de um projeto de software revela-se como um fator deveras limitante.

Após uma fase inicial em que se apresentam as expectativas e em que se estabelece o escopo da aplicação a ser gerada, a área que solicitou um sistema apenas terá uma visão do resultado ao término do projeto. Mudanças eventuais em requisitos ou, mesmo, em regras que definem o comportamento destes podem prejudicar todo um esforço de meses. Tais alterações podem ainda comprometer um orçamento previamente acordado, além de não ser raro que o produto resultante sequer chegue a ser utilizado (por estar distante daquilo que se aguardava).

Conforme já frisado neste artigo, o cotidiano dinâmico de muitas organizações acabaria impondo o surgimento de um novo enfoque para a construção de softwares. Com base em uma série de parâmetros pré-estipulados, mas que também se encontram sujeitos a mudanças ao longo de um projeto, outras abordagens foram desenvolvidas de forma a tornar o desenvolvimento de aplicações mais flexível e produzindo resultados mais próximos do esperado.

Procurando fazer um contraponto às deficiências do modelo cascata, a metodologia RUP (Rational Unified Process) foi criada para permitir um desenvolvimento incremental e com entregas sucessivas de partes do software combinado. Amparando-se em conceitos da Orientação a Objetos, além de representação de estruturas de sistemas por meio da linguagem UML.

RUP é uma boa alternativa para projetos de grande porte que exigem um processo bem estruturado e que prime por uma documentação rica em detalhes. Esta é uma demanda bastante comum em organizações sujeitas a rigorosos procedimentos de auditoria. Este modelo está dividido em fases de análise (Concepção), modelagem e arquitetura do sistema a ser entregue (Elaboração), implementação (Construção) e Transição (implantação), sendo possível ainda que as diferentes atividades destas etapas possam vir a acontecer de maneira paralela em determinados momentos.

Ainda sobre o RUP, esta metodologia é na verdade uma implementação de um processo conhecido como Unified Process, representando uma solução específica da IBM para a condução de atividades voltadas ao desenvolvimento de software. É importante destacar também que existe uma série de ferramentas disponibilizadas para a geração dos diversos artefatos previstos por este modelo.

Nem sempre um processo trabalhoso e extremamente detalhado como o RUP pode ser a melhor alternativa em projetos de software. Equipes de tamanho reduzido, projetos que não são extensos e requisitos mudando constantemente motivariam a busca por novas formas de se controlar o processo de desenvolvimento de uma aplicação. Essas necessidades levariam, conseqüentemente, ao desenvolvimento de uma série de padrões que compõem um agrupamento de técnicas conhecidas como metodologias ágeis.

Em 2001 seria publicado, a partir do trabalho conjunto de diversos especialistas da área de sistemas, o Manifesto para Desenvolvimento Ágil de Software. Tal declaração enfatiza a entrega de software operável, atuando em conjunto com o solicitante do mesmo, além de frisar a necessidade de uma interação adequada entre os diversos envolvidos num projeto e a flexibilidade diante de mudanças. São doze os princípios que norteiam o Manifesto Ágil (ou em inglês “Agile Manifesto”):

Satisfação das expectativas da área solicitante/cliente;

Uma postura mais positiva diante da necessidade de mudanças, procurando inclusive obter benefícios e explorar oportunidades não vislumbradas antes;

Entregas mais frequentes, a fim de possibilitar que o cliente possa acompanhar melhor a evolução do projeto, visando assegurar que os requisitos esperados realmente têm sido atendidos;

A colaboração entre todos aqueles que participam do projeto, quer sejam pessoas da área que solicitou a aplicação ou, mesmo, desenvolvedores focados em atividades de implementação, de forma a se agir como um time coeso;

Assegurar que existam condições para que todos possam trabalhar motivados;

Transmissão de informações através de conversas face-a-face;

Garantir que o software é entregue funcionando;

Manter um ritmo constante de trabalho, de forma que o processo como um todo continue de um modo sustentável, sem grandes percalços;

Forte ênfase na qualidade do que é produzido;

Foco na simplicidade;

Equipes auto-organizáveis, capazes de tomar decisões técnicas que viabilizem a evolução contínua do processo em que se encontram inseridas;

Uma reflexão constante dos rumos que estão sendo trilhados, a fim de se aumentar a eficácia da equipe, bem como ajustar o comportamento da mesma para com metas estabelecidas previamente.

Além do Scrum que será descrito em maiores detalhes nas próximas seções, outros modelos ágeis que desfrutaram de uma boa aceitação na área de desenvolvimento de software são as metodologias XP e Lean IT.

XP (abreviação do inglês “Extreme Programming”) é uma metodologia de desenvolvimento de software que se caracteriza por uma forte ênfase na elaboração e execução contínua de testes unitários, bem como pela codificação em pares: uma dupla de desenvolvedores participa da implementação de uma ou mais funcionalidades, sendo que isto acontece em torno de um único computador, com a pessoa responsável por escrever o código sendo auxiliada pelo parceiro que a observa e a orienta conforme a evolução das atividades. Esta abordagem em que dois profissionais interagem em conjunto procura diminuir a incidência de falhas, assim como possibilitar uma melhor qualidade do código resultante.

Lean IT é uma metodologia ágil baseada no modelo de gestão de produção da montadora japonesa Toyota. Este método prioriza a organização das atividades em uma maneira na qual se elimine ou reduza a perda de recursos (sobretudo tempo), de forma a tornar a aumentar a eficiência dos processos e atender mais rapidamente às necessidades das áreas que solicitaram um projeto.

Independentemente da opção escolhida para gerenciar projetos dentro da área de Tecnologia, é necessário sempre ter em mente que cada padrão pode ser adaptado ou, até mesmo, combinado a outra abordagem, de maneira a se adaptar melhor às necessidades da organização.

Métodos ágeis não estão restritos a uma tecnologia (.NET, Delphi, Java etc.); na verdade, as diversas técnicas existentes procuram ser a base para uma mudança de postura rumo a um melhor gerenciamento das atividades cotidianas.

Desenvolvimento ágil com Scrum

Scrum é uma metodologia ágil voltada ao desenvolvimento de software. Surgido ainda na década de 1990, este modelo é resultado dos esforços conjuntos de especialistas da área de sistemas, com destaque especial para Ken Schwaber e Jeff Sutherland. O termo “scrum” é originário do meio esportivo: no jogo de rugby esta palavra de língua inglesa refere-se ao reinício de uma partida logo após uma infração leve. Na Figura 1 é apresentada a representação esquemática de um processo baseado nesta metodologia, com os diferentes itens citados sendo descritos nas próximas seções.

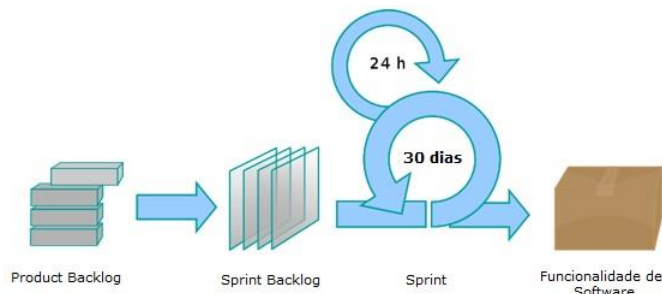


Figura 1: Visão geral dos prRepresentação lúdica do comprometimento com um projeto Scrum

Assim como outros métodos com um enfoque similar, a proposta do Scrum é fornecer subsídios para o gerenciamento de atividades muitas vezes complexas, porém de uma forma flexível e que facilite a adaptação do projeto diante das inevitáveis mudanças. São três as ideias principais em que a metodologia Scrum se ampara:

Transparência;

Inspeção;

Adaptação.

A noção de transparência deve ser compreendida como a existência de um consenso mútuo entre todos os participantes de um projeto, considerando para isto termos específicos de negócio, regras que caracterizam processos e outros aspectos. Inclusive o significado de "pronto", algo tão controverso em projetos de software que apresentam deficiências, está contemplado dentro do conceito de transparência.

Outro ponto importante da metodologia Scrum é a atividade de inspeção. A verificação contínua do que é produzido é fator determinante para que se tenha a certeza de que o projeto caminha dentro do estipulado, bem como para diagnosticar desvios indesejáveis e atuar de forma corretiva sobre estes últimos.

Já o conceito de adaptação vai de encontro a um dos principais objetivos dos métodos ágeis: a flexibilidade diante de mudanças. Não serão raras as ocasiões em que alterações em demandas de clientes ou ainda, regulamentações governamentais e de ordem setorial, acarretarão desvios de rotas pré-estabelecidas. Diante disto, ajustes se fazem necessários, atenuando desse modo outros efeitos indesejáveis em momentos posteriores.

Os diferentes papéis de atuação na metodologia Scrum

Um conjunto de papéis bem definidos determina, através de atribuições, como será a atuação dos diversos profissionais que participarão de um projeto baseado na metodologia Scrum:

Product Owner: representante da área cliente/solicitante;

Scrum Master: o líder que gerenciará o projeto;

Equipe de Desenvolvimento/Time: os profissionais responsáveis pela criação do produto esperado.

O Product Owner é o ponto que centraliza a interação com a área/grupo de usuários que solicitou a execução de um projeto. A partir do mesmo são definidas prioridades, o que deverá ou não ser implementado e a validação dos diferentes resultados ao longo do processo de desenvolvimento.

Já o Scrum Master corresponde ao papel do Gerente de Projetos tradicional. Além de ser um facilitador removendo impedimentos e um mediador em prováveis conflitos, este profissional garante que a equipe sob sua supervisão siga de maneira adequada as práticas de Scrum.

Por fim a Equipe de Desenvolvimento (ou Time) elabora estimativas, estipula tarefas, implementa o produto dentro de níveis de qualidade pré-estipulados e cuida da apresentação do mesmo ao cliente/solicitante. Conforme já mencionado anteriormente, espera-se que tal equipe conte com um caráter auto-gerenciável, com o comprometimento e uma postura multifuncional dos membros representando um fator crucial para o sucesso do projeto.

Além dos três papéis já descritos, certamente também existirão envolvidos com o projeto, mas que não desempenham um papel direto na sua execução. Estes elementos podem englobar usuários, gerentes, diretores ou departamentos que possuem interesses (neste caso são conhecidos dentro da Gerência de Projetos como “stakeholders”) ou ainda, são afetados pelos resultados do produto final.

Considerando tudo isto, criou-se uma forma lúdica de representar o grau de comprometimento de uma pessoa em um projeto que segue as práticas do Scrum; para isto, foi utilizada uma sequência de histórias em que interagem um porco e um frango (Figura 2). Porcos representariam profissionais realmente engajados/comprometidos com o sucesso do projeto (Product Owner, Scrum Master, Time), ao passo que frangos seriam pessoas relacionadas indiretamente e sem uma maior disposição para com o mesmo (usuários comuns, gerentes ou áreas afetadas).



Figura 2: Representação lúdica do comprometimento com um projeto Scrum

OBSERVAÇÃO: No site “Implementing Scrum” (endereço indicado na seção de Links deste artigo) podem ser encontradas outras tiras que descrevem de forma bem humorada a metodologia Scrum, assim como problemas do dia-a-dia e os esforços na condução de projetos sob este padrão.

Eventos possíveis em Scrum

Como outros métodos ágeis, Scrum é uma metodologia que prima pelo desenvolvimento iterativo e incremental de software. Em termos práticos, isto significa que ciclos contendo um conjunto de específico de atividades são repetidos continuamente ao longo de um projeto; por incremental, deve-se ter em mente a idéia de sucessivas entregas de funcionalidade, acrescentando aquilo que se espera do software em intervalos constantes de tempo.

Antes de prosseguir com a descrição dos diferentes eventos de Scrum, faz-se necessário conceituar as idéias de Time-Box e Sprint. Uma Time-Box nada mais é do que a quantidade de tempo estipulado para a realização de uma iteração. Esta última recebe o nome de Sprint, sendo que essa estimativa de tempo não pode ser alterada, a fim de com isto garantir a entrega sem atrasos e facilitar o planejamento. Diante da possibilidade de erros na estimativa, deve-se proceder com uma redução do escopo, sem contudo afetar substancialmente as metas da Sprint ou obrigar a um aumento na quantidade de horas ou dias planejados. O comum é que uma Sprint dure de duas a quatro semanas.

A partir destas explanações, são possíveis dentro de Scrum os seguintes eventos:

Reunião de Planejamento da Sprint;

Reunião Diária;

Revisão da Sprint;

Retrospectiva da Sprint.

A Reunião de Planejamento da Sprint é uma atividade com duração de 8 horas e da qual participam todos os profissionais comprometidos com o projeto (Product Owner, Scrum Master, Equipe de Desenvolvimento). Esta reunião é formada por duas etapas: num primeiro momento o Product Owner define a prioridade de funcionalidades a serem implementadas (a partir do Backlog); posteriormente, a Equipe montará sua própria lista de trabalho (Sprint Backlog) com base no que foi exposto pelo Product Owner.

Já a Reunião Diária é uma curta sessão de 15 minutos, em que membros da Equipe e o Scrum Master comentam a situação atual (muitas vezes em pé dentro de um recinto). Isto normalmente envolve uma rápida explanação do que foi feito no dia anterior, o que será realizado na data atual e uma discussão de prováveis impedimentos que foram encontrados.

A Revisão da Sprint é uma reunião de normalmente 4 horas, em que estão presentes o Product Owner, o Scrum Master e a Equipe (pode acontecer ainda de outras pessoas serem convidadas para participar). As atividades desempenhadas durante a Sprint são apresentadas, procedendo-se com a entrega do software funcionando (conforme pregado pela metodologia).

Por fim, há ainda a Retrospectiva da Sprint. Trata-se de uma reunião de geralmente 3 horas entre Equipe e Scrum Master. Nesta discussão aborda-se o que deu certo e aquilo que falhou, além de se estudarem formas para se melhorar num próximo ciclo (Sprint).

Artefatos de Scrum

Por mais que como uma metodologia ágil Scrum priorize a entrega do software em detrimento de uma extensa e trabalhosa documentação, a elaboração e a consequente manipulação de alguns artefatos neste modelo é de fundamental importância para o controle das atividades rotineiras. A seguir estão listados tais documentos:

Backlog do Produto (ou em inglês “Product Backlog”);

User Story;

Sprint Backlog;

Gráfico de Burn Down.

O Product Backlog é uma listagem que contempla todas as funcionalidades desejadas para o software que se está implementando. Além disso, as informações contidas neste tipo de controle podem conter ainda uma ordem de prioridade, sendo incumbência do Product Owner criar e controlar o status dos diferentes elementos do Backlog.

Uma User History é uma pequena história que descreve as características esperadas para uma funcionalidade constante no Backlog do Produto. Constam no documento que representa tal história um título, uma descrição clara do que se necessita, bem como é possível se indicar ainda uma prioridade para execução da tarefa.

A Sprint Backlog é uma relação de tarefas elaborada pelo Time de Desenvolvimento durante a segunda etapa da Reunião de Planejamento da Sprint. Trata-se de algo que está em conformidade com o conceito de equipes auto-gerenciáveis, uma vez que os profissionais responsáveis por isto planejam como será o dia-a-dia de desenvolvimento a partir das prioridades apontadas pelo Product Owner.

O Gráfico de Burn Down é uma ferramenta de gerenciamento. Este artefato costuma ser atualizado diariamente, servindo de base para a comparação entre o que foi planejado e aquilo que realmente se realizou. Pode ser considerado um instrumento para tomada de decisão, uma vez que fornece

PMBOK: Conceitos

Principais conceitos do guia PMBOK

O objetivo deste artigo é realizar uma introdução sobre PMBOK, evidenciando os principais conceitos apresentados no guia.

O PMBOK

O PMBOK (Project Management Book Of Knowledge ou Guia do Conhecimento em gerenciamento de Projetos), como o nome já diz, é um guia criado pelo PMI (Project Management Institute). O guia tem o objetivo de divulgar boas práticas que podem ser aplicadas em gerenciamento de projetos.

Além de definir boas práticas, o PMBOK também faz uma introdução aos principais conceitos no campo de gerenciamento de projetos.

Conceitos Importantes Para Entender O Guia

Formalmente, **PMBOK** pode ser definido como uma norma reconhecida para a profissão de gerenciamento de projetos que identifica o subconjunto do conjunto de conhecimentos em gerenciamento amplamente reconhecido como boa prática. **Amplamente reconhecido** significando que o conhecimento e as práticas descritas são aplicáveis à maioria dos projetos na maior parte do tempo e que existe um consenso em relação ao seu valor e utilidade. **Boa prática** significa que há um consenso geral de que a aplicação correta dessas habilidades, ferramentas e técnicas pode aumentar as chances de sucesso em uma ampla gama de projetos.

Outra forma de definir PMBOK formalmente seria: PMBOK é o padrão para gerenciar a maioria dos projetos na maior parte das vezes em vários tipos de setores da indústria. Ele descreve os processos, ferramentas e técnicas de gerenciamento de projetos usados até a obtenção de um resultado bem sucedido.

Então o PMI sugere que o PMBOK deve ser seguido a risca para todos os projetos?

Não. Uma boa prática não quer dizer que o conhecimento descrito deva ser obrigatoriamente aplicado. Cada projeto deve ser avaliado para decidir quais práticas devem ser aplicadas.

O guia também define **projeto** como um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. A principal característica de um projeto está em sua natureza temporária. Devido a esta característica o projeto possui um início e um fim. Além disto, vale a pena ressaltar que, apesar de temporário, um projeto pode ter uma longa duração. Por exemplo, um projeto pode durar diversos. Mas há sempre uma data programada para o início e uma data programada para o fim.

Depois de definir projeto, é possível entender melhor o que seria gerenciamento de projetos. O PMBOK define **gerenciamento de projetos** como a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto a fim de atender aos seus requisitos.

No PMBOK, o gerenciamento de um projeto é realizado aplicando-se 42 processos que são organizados em 5 grupos de processos. Esta estrutura do PMBOK será vista adiante. Por hora, é preciso saber que o gerenciamento de projetos no PMBOK inclui:

§ Identificação dos Requisitos

§ Adaptação às diferentes necessidades, preocupações, e expectativas das partes interessadas à medida que o projeto é planejado e realizado

§ Balanceamento das restrições conflitantes do projeto que incluem, mas não se limitam a:

- * Escopo
- * Qualidade
- * Cronograma

* Orçamento

* Recursos

* Risco

Apesar do PMBOK não contemplar o gerenciamento de portfólios e programas, esses conceitos são apresentados no guia, pois são importantes para o entendimento de gerenciamento de projetos. Pelo mesmo motivo, o conceito de gerenciamento de portfólios e gerenciamento de programas também são apresentados.

Um **portfólio** refere-se a um conjunto de projetos ou programas e outros trabalhos, agrupados para facilitar o gerenciamento eficaz desse trabalho a fim de atingir os objetivos de negócios estratégicos.

O **gerenciamento de portfólios** se refere ao gerenciamento centralizado de um ou mais portfólios, que inclui identificação, priorização, autorização, gerenciamento e controle de projetos, programas e outros trabalhos relacionados, para atingir objetivos de negócios estratégicos específicos.

Um **programa** é definido como um grupo de projetos **relacionados** gerenciados de modo coordenado para obtenção de benefícios e controle que não estariam disponíveis se eles fossem gerenciados individualmente. É importante ressaltar que um projeto pode não fazer parte de um programa, mas um programa sempre terá projetos.

O **gerenciamento de programas** é definido como o gerenciamento centralizado e coordenado de um programa para atingir objetivos e benefícios do mesmo.

A principal **diferença entre um programa e um portfólio** é que em um programa os projetos estão relacionados através do resultado comum ou da capacidade coletiva. Se a relação entre os projetos for apenas um cliente, vendedor, tecnologia ou recurso compartilhado, o esforço deve ser gerenciado como um portfólio de projetos e não como um programa.

Bons **exemplos de programas** são programas de energia do governo que inclui diversos projetos que têm a finalidade de produzir energia. Por exemplo, projeto para construção de uma usina nuclear, projeto para construção de uma usina hidrelétrica, projeto para fornecimento da energia produzida. Todos esses projetos podem ser gerenciados como um programa.

Então tudo pode ser gerenciado como um projeto?

A resposta é não!

Também existem as **operações** que são uma função organizacional que realiza a execução contínua de atividades que produzem o mesmo produto ou fornecem um serviço repetitivo (Operações de contabilidade, fabricação e produção). Portanto, como são contínuas ferem a principal característica de um projeto de ser temporário.

O Guia PMBOK também define qual é o **papel de um gerente de projetos**. Segundo o guia, o gerente de projetos é a pessoa designada pela organização executora para atingir os objetivos do projeto. O gerente de projetos deve ter três características:

§ **Conhecimento:** refere-se ao que o gerente de projetos sabe sobre gerenciamento de projetos

§ **Desempenho:** refere-se ao que o gerente de projetos é capaz de realizar enquanto aplica seu conhecimento em gerenciamento de projetos

§ **Pessoal:** refere-se ao comportamento do gerente na execução do projeto ou de atividade relacionada.

O último conceito apresentado na primeira parte do guia se refere à **fatores ambientais da empresa**.

Os **fatores ambientais** de uma empresa refere-se tanto aos fatores ambientais internos e externos que cercam ou influenciam o sucesso de um projeto. Esses fatores podem aumentar ou restringir as opções de gerenciamento de projetos e podem ter uma influência positiva ou negativa no resultado.

PMBOK e Gerenciamento de Projetos

Gerenciamento de projetos (GP) é uma área de atuação e conhecimento que tem ganhado, nos últimos anos, cada vez mais reconhecimento e importância. Um dos principais difusores do gerenciamento de projetos e da profissionalização do gerente de projetos é o Instituto de Gerenciamento de Projetos (PMI - Project Management Institute).

Fundado nos Estados Unidos em 1969, o PMI é uma associação profissional mundialmente difundida, atualmente com meio milhão de membros em mais de 180 países. O PMI é distribuído geograficamente pelo mundo em Capítulos. Existe o PMI Brasil - Integração Nacional, programa dos capítulos do PMI em diversos estados brasileiros.

Duas das principais iniciativas do PMI na difusão do conhecimento em gerenciamento de projetos são as certificações profissionais em gerência de projetos — Project Management Professional (PMP) e Certified Associate in Project Management (CAPM) — e a publicação de padrões globais de gerenciamento de projetos, programas e portfólio, sendo a mais popular delas o **Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos (Guia PMBOK® - Project Management Body of Knowledge)**.

Editado na forma de livro, o Guia PMBOK está atualmente na quinta edição de dezembro de 2012 e traduzido oficialmente para diversos idiomas, inclusive o português do Brasil. As edições anteriores foram publicadas nos anos de 1996, 2000, 2004 e 2008.

O Guia PMBOK é reconhecido como um Padrão Nacional Americano pelo ANSI. A quinta edição é o padrão ANSI/PMI 99-001-2013 e teve alinhamento com a norma internacional ISO 21500:2012 (também disponível como norma brasileira ABNT NBR ISO 21500:2012) - Orientações sobre Gerenciamento de Projetos, lançada pela ISO em setembro de 2012 visando unificar e criar normas que deverão ser seguidas mundialmente.

O Guia PMBOK formaliza diversos conceitos em gerenciamento de projetos, como a própria definição de projeto e do seu ciclo de vida. Também identifica na comunidade de gerenciamento de projetos um conjunto de conhecimentos amplamente reconhecido como boa prática, aplicáveis à maioria dos projetos na maior parte do tempo. Estes conhecimentos estão categorizados em dez áreas e os processos relacionados são organizados em cinco grupos ao longo do ciclo de vida do projeto.

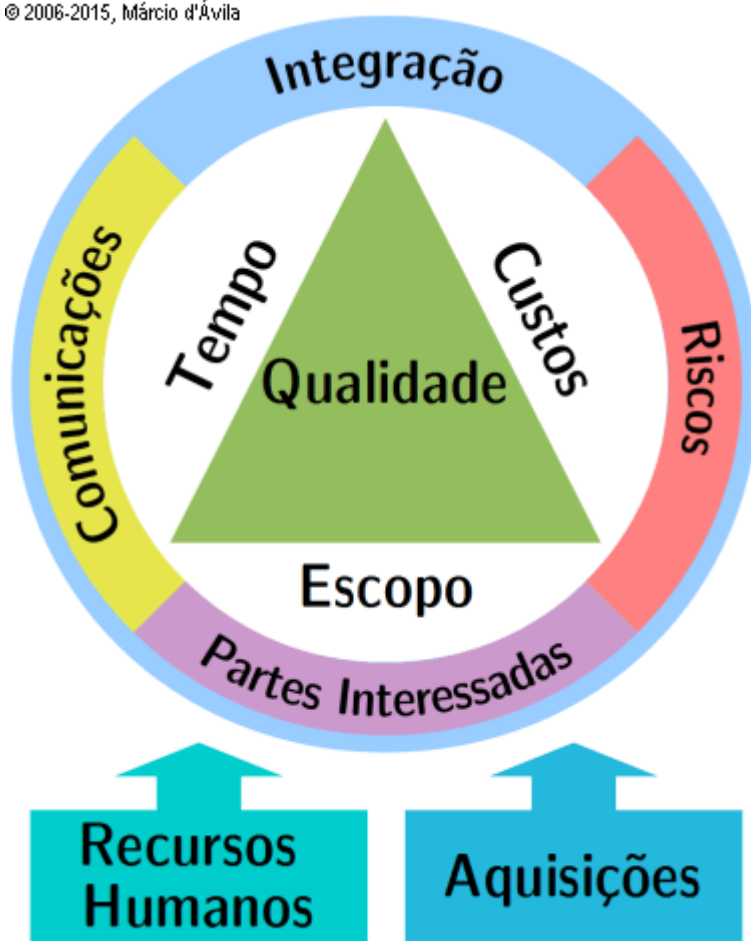
Projetos E Seu Gerenciamento

Um **projeto** é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. [PMI 2012, p. 3]

Dois termos da definição de projetos merecem destaque. Temporário não significa necessariamente de curta duração, mas sim que um projeto possui um início e um término definidos. Isso distingue o projeto dos trabalhos operacionais de natureza contínua.

E exclusivo indica a singularidade da natureza de cada projeto, pois mesmo que elementos repetitivos ou similares possam estar presentes em algumas entregas do projeto, o resultado de cada projeto é obtido sob uma combinação exclusiva de objetivos, circunstâncias, condições, contextos, fornecedores etc.

© 2006-2015, Márcio d'Ávila



Gerenciamento de projetos é a aplicação de conhecimentos, habilidades, ferramentas e técnicas adequadas às atividades do projeto, para atender aos seus requisitos. [PMI 2012, p. 5]

Áreas de conhecimento

As dez **áreas de conhecimento** caracterizam os principais aspectos envolvidos em um projeto e no seu gerenciamento:

- Integração
- Escopo
- Tempo
- Custos
- Qualidade
- Recursos humanos
- Comunicações
- Riscos
- Aquisições
- Partes interessadas

Escopo, Tempo, Custos e Qualidade são os principais determinantes para o objetivo de um projeto:

entregar um resultado de acordo com o escopo, no prazo e no custo definidos, com qualidade adequada; em outras palavras, o que, quando, quanto e como. Recursos Humanos e Aquisições são os insumos para produzir o trabalho do projeto. Comunicações, Partes interessadas e Riscos devem ser continuamente tratados para manter as expectativas e as incertezas sob controle, assim como o projeto no rumo certo. E Integração abrange a orquestração de todos estes aspectos.

Partes Interessadas passou a ser área de conhecimento no PMBOK 5ª Edição (2012), resultante de um desdobramento de Comunicações.

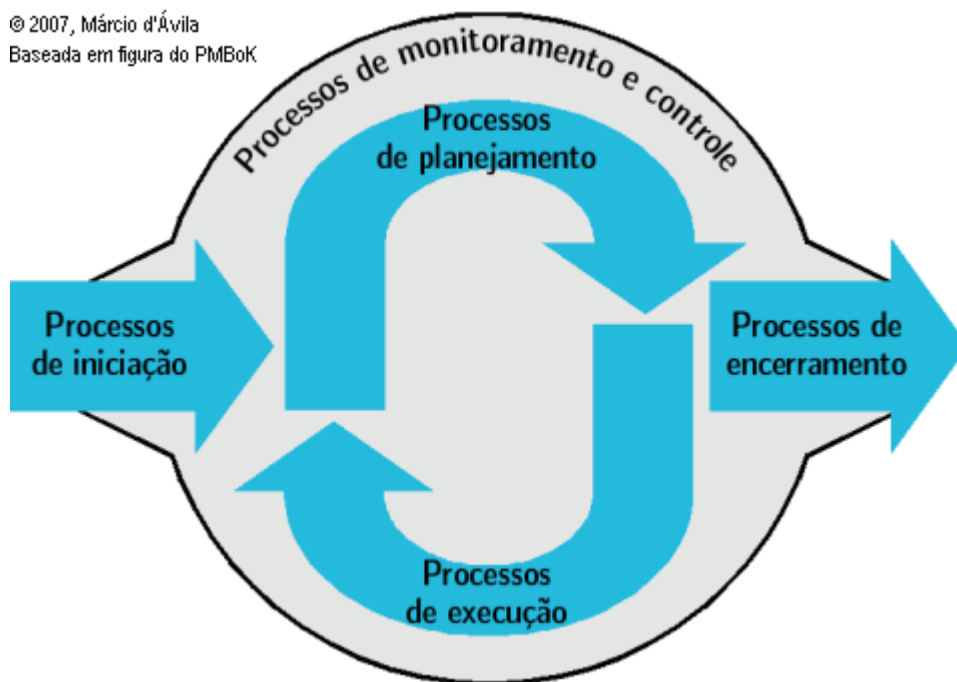
Um projeto consiste nisso: pessoas (e máquinas) que utilizam tempo, materiais e dinheiro realizando trabalho coordenado para atingir determinado objetivo.

Processos Do Gerenciamento De Projetos

A aplicação dos conhecimentos requer a adoção eficaz de processos apropriados. Cada área de conhecimento abrange diversos processos no gerenciamento de projetos.

Um **processo** é um conjunto de ações e atividades interrelacionadas que são executadas para alcançar um objetivo. Cada processo é caracterizado por suas entradas, ferramentas e técnicas que podem ser aplicadas, e as saídas resultantes. [PMI 2012, p. 47]

© 2007, Márcio d'Ávila
Baseada em figura do PMBoK



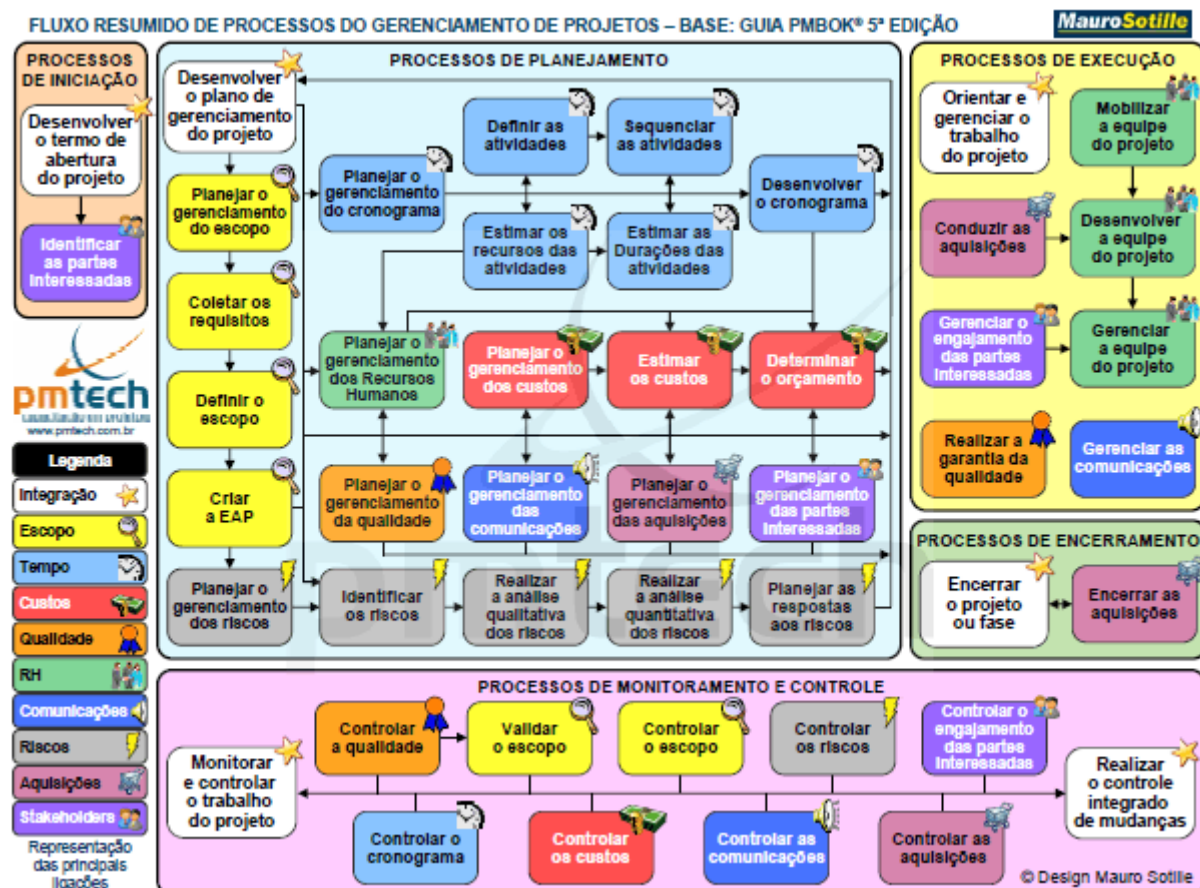
Os cinco **grupos de processos** de gerenciamento de projetos são:

1. Iniciação
2. Planejamento
3. Execução
4. Monitoramento e Controle
5. Encerramento

Os grupos de processos de gerenciamento de projetos têm grande correspondência com o conceito do Ciclo PDCA (Plan - Do - Check - Act): Planejar - Fazer - Verificar - Agir (corrigir e melhorar). O grupo de Planejamento corresponde ao Planejar; Execução, ao Fazer; e Monitoramento e controle englobam Verificar e Agir. E como a natureza dos projetos é finita, o PMBOK ainda caracteriza os grupos de processos que iniciam (Iniciação) e finalizam (Encerramento) um projeto.

Além de conceituar os aspectos fundamentais do gerenciamento de projetos, de forma a promover um vocabulário comum dentro dessa profissão, o Guia PMBOK documenta (define e descreve) processos de gerenciamento de projetos e os apresenta didaticamente, organizados em um capítulo por área de conhecimento. Em cada processo, são abordados suas entradas e saídas, suas características, bem como os artefatos, técnicas e ferramentas envolvidas.

O excelente diagrama com um fluxo proposto por Mauro Sotille, disponível nas seções de templates e artigos sobre Gerenciamento de Projetos do portal da empresa PM Tech, relaciona de forma gráfica e sintética todos os **47 processos de gerenciamento de um projeto** descritos no PMBOK 5ª Edição, indicando também os cinco grupos em que os processos se distribuem e as respectivas áreas de conhecimento associadas a cada um.



Crédito/Fonte: Fluxo de Processos do Gerenciamento de Projetos - PMBOK 5ª Edição [PDF], por Mauro Afonso Sotille, PM Tech - Capacitação em Gerenciamento de Projetos, Porto Alegre - RS. Disponível também Fluxo de Processos do GP - PMBOK 4ed e Visão Geral dos Processos do GP - PMBOK 4ed.

Para estes mesmos 47 processos de gerenciamento de projetos do PMBOK 2012, a matriz a seguir provê uma visão quantitativa de sua distribuição pelas áreas de conhecimento e pelos grupos de processos. Clique na figura para exibir uma descrição resumida dos respectivos processos. Veja também Grupos de Processos e Áreas de Conhecimento - PMBOK 5ed [PDF], por Mauro Sotille, PM Tech.

	Iniciação	Planejamento	Execução	Controle	Encerramento	Σ
Escopo		4		2		6
Tempo		6		1		7
Custos		3		1		4
Qualidade		1	1	1		3
Recursos Humanos		1	3			4
Partes Interessadas	1	1	1	1		4
Aquisições		1	1	1	1	4
Comunicações		1	1	1		3
Riscos		5		1		6
Integração	1	1	1	2	1	6
© 2006-2015, Márcio d'Ávila	2	24	8	11	2	47

Pelo diagrama é fácil perceber algumas características lógicas dos processos de gerenciamento de um projeto:

- praticamente todas as áreas de conhecimento são abordadas nas atividades de Planejamento (definir, estimar e planejar cada aspecto) e de Monitoramento e Controle (controlar) — no PMBOK 4ª edição, o processo de Gerenciar a equipe passou ao grupo de Execução, deixando apenas a área de RH sem processos no grupo de Controle;
- quanto ao Planejamento, os aspectos envolvidos mais amplamente são tempo, riscos, escopo e custos;
- quanto à Execução, os aspectos envolvidos mais ativamente são a equipe (RH), as comunicações, as partes interessadas, as aquisições, e a garantia da qualidade;
- a integração se faz presente em todos os momentos do projeto.
- na figura com as descrições, os grupos de processos representam os tipos de atividades, as áreas de conhecimento caracterizam os assuntos, e seu cruzamento induz, de forma bastante intuitiva, os respectivos verbos — definir, planejar, estimar, gerenciar, monitorar, controlar, encerrar etc. — e substantivos que descrevem os processos de gerenciamento relacionados.

Isso mostra que os conceitos e melhores práticas que o PMBOK reúne, organiza e formaliza estão naturalmente presentes na essência do gerenciamento de qualquer bom projeto.

O Gerente De Projetos

O **gerente do projeto** é a pessoa designada pela organização responsável pela condução do projeto, com a missão de zelar para que os objetivos do projeto sejam atingidos. O gerente de projetos tem sido caracterizado por um perfil profissional com domínio e experiência especializados nos processos e nas áreas de conhecimento do gerenciamento de projetos.

O trabalho do gerente de um projeto pode ser sintetizado em dois grandes elementos:

- **Planejar** (antes) e **Controlar** (durante) as atividades do projeto e seu gerenciamento, conforme se pode constatar pela concentração de processos de gerenciamento de um projeto abrangendo todas os aspectos envolvidos.
- **Comunicar**: os gerentes de projetos passam a maior parte do seu tempo tratando com os membros da equipe e outras partes interessadas do projeto.

Para obter eficácia no gerenciamento, em especial na comunicação, os gerentes de projetos devem dominar habilidades interpessoais. Isso significa um longo e contínuo processo de crescimento pessoal e desenvolvimento gerencial.

Segundo a psicóloga Fela Moscovici [Moscovici 1981], **competência interpessoal** é a habilidade de lidar eficazmente com relações interpessoais, de lidar com outras pessoas de forma adequada às necessidades de cada um e às exigências da situação.

Competência interpessoal é resultante de percepção acurada e realística das situações interpessoais e de habilidades comportamentais específicas que conduzem a consequências significativas no relacionamento duradouro e autêntico, satisfatório para as pessoas envolvidas. Um terceiro componente dessa competência refere-se ao relacionamento em si, e compreende a dimensão emocional-afetiva, predominantemente.

Lidar com situações interpessoais requer sensibilidade e flexibilidade perceptiva e comportamental, que significa procurar ver vários ângulos ou aspectos da mesma situação e atuar de forma diferenciada, não rotineira, experimentando novas condutas percebidas como alternativas de ação.

Destacam-se as seguintes **habilidades interpessoais** para o gerente de projetos:

- Comprometimento, responsabilidade, ética e honestidade;
- Transparência, franqueza, clareza e objetividade;
- Liderança, agregação, motivação e entusiasmo;
- Solução de conflitos e problemas;
- Negociação, influência e persuasão;
- Decisão, iniciativa e proatividade;
- Organização e disciplina;
- Autocontrole, equilíbrio e resiliência;
- Empreendedorismo;
- Eficácia.

O PMI mantém um Código de Ética e Conduta Profissional (Project Management Institute Code of Ethics and Professional Conduct), criado para incutir confiança à profissão de gerenciamento de projetos e auxiliar os praticantes a se tornarem melhores profissionais. Para isso, o código descreve as expectativas que os profissionais de gerenciamento de projetos tem de si e de seus colegas. Ele exige que os profissionais demonstrem compromisso com a conduta ética e profissional, sendo específico quanto à obrigação básica de responsabilidade, respeito, justiça e honestidade. Isso inclui respeitar as leis, regulamentos e políticas organizacionais e profissionais.

Mais que ser um facilitador, o gerente de projetos deve fazer a diferença no bom andamento e no sucesso dos projetos.

Partes Interessadas

Partes interessadas, intervenientes ou — do termo em inglês — **stakeholders** são pessoas, grupos ou organizações que podem afetar, serem afetados ou sentirem-se afetados por uma decisão,

atividade ou resultado de um projeto. Elas englobam os envolvidos no projeto, ou cujos interesses podem ser positiva ou negativamente afetados pela execução ou pelos resultados do projeto. Elas também podem exercer influência sobre o projeto e suas saídas. [PMI 2012, p. 394]

Desde a iniciação do projeto, a equipe de gerenciamento precisa identificar as partes interessadas internas e externas. Ao longo do planejamento e da execução do projeto, o gerente do projeto e sua equipe devem gerenciar as diferentes necessidades, preocupações e expectativas das partes interessadas, bem como a influência destas no projeto, para garantir um resultado bem sucedido.

Alguns exemplos de possíveis partes interessadas podem incluir:

- **Patrocinador** (Sponsor): pessoa ou grupo que fornece os recursos financeiros para a realização do projeto, e que também provê o aval estratégico e político que viabiliza e promove o projeto e o defende;
- **A equipe do projeto**, que inclui o gerente do projeto, a equipe de gerenciamento do projeto, e outros membros da equipe que executam trabalho no projeto mas não estão necessariamente envolvidos com o gerenciamento; [PMI 2008, p. 26]
- Clientes e usuários;
- Presidente, donos e executivos;
- Acionistas e investidores;
- Gerentes funcionais;
- Escritório de projetos (Project Management Office - PMO), gerentes e comitês de portfólios e de programas;
- Fornecedores e parceiros comerciais;
- Concorrentes;
- Governo, em suas diversas esferas e poderes;
- Organismos de regulação e fiscalização internos e externos, incluindo auditorias, agências, conselhos, sindicatos e associações institucionais, profissionais e oficiais;
- Organizações não governamentais (ONG);
- Comunidades, vizinhança e população abrangida pelas ações e resultados do projeto.

Outros elementos importantes que influenciam projetos são as **culturas e estilos organizacionais**, bem como os **fatores ambientais** da empresa, do mercado, da sociedade e da localização geopolítica onde o projeto acontece.

Projetos, Programas E Portfólios

O ecossistema de gerenciamento de projetos está inserido em um contexto mais amplo, regido pelo gerenciamento de programas e gerenciamento de portfólios, principalmente em organizações mais maduras em lidar com projetos. As estratégias, prioridades e o planejamento organizacional impactam a priorização de projetos com base em risco, financiamento e no plano estratégico da organização.

O Guia PMBOK define um **programa** como um grupo de projetos (bem como subprogramas e atividades de programa) relacionados, gerenciados de modo coordenado visando a obtenção de benefícios que não estariam disponíveis se eles fossem gerenciados individualmente. Um programa pode incluir elementos de trabalho fora do escopo dos projetos distintos nele. [PMI 2012, p. 9]

Um projeto pode ou não fazer parte de um programa, mas um programa sempre terá projetos. Enquanto projetos possuem objetivos definidos e específicos, programas possuem um escopo maior e visam benefícios mais significativos.

Gerenciamento de programas é definido como o gerenciamento centralizado e coordenado de um programa para atender seus requisitos e obter benefícios e controle que transcendem os projetos individualmente.

Já um **portfólio** refere-se a um conjunto de projetos ou programas e outros trabalhos, agrupados para facilitar o gerenciamento eficaz desse trabalho a fim de atingir os objetivos estratégicos de negócios. Os projetos ou programas do portfólio podem não ser interdependentes ou diretamente relacionados. [PMI 2008, p. 8]

O **gerenciamento de portfólios** é o gerenciamento centralizado de um ou mais portfólios, que inclui identificação, priorização, autorização, gerenciamento e controle de projetos, programas e outros trabalhos relacionados, para atingir objetivos estratégicos específicos de negócios. Ainda segundo o PMBOK, o gerenciamento de portfólios se concentra em garantir que os projetos e programas sejam analisados a fim de priorizar a alocação de recursos, e que o gerenciamento do portfólio seja consistente e esteja alinhado às estratégias organizacionais. [PMI 2008, p. 9]

Portfólios possuem um escopo de negócios no ambiente mais amplo da organização, alinhados com objetivos e prioridades estratégicas de negócios. Um portfólio de nível mais elevado pode ser composto de portfólios de nível mais específico, programas e projetos.

Os projetos, em programas ou portfólios, são frequentemente utilizados como meio de atingir metas e objetivos organizacionais, geralmente no contexto de um planejamento estratégico. Os projetos são normalmente autorizados como resultado de uma ou mais das seguintes considerações estratégicas: [PMI 2008, p. 10]

- Demanda de mercado;
- Oportunidade ou necessidade estratégica de negócios;
- Solicitação de cliente;
- Avanço tecnológico;
- Requisito legal.

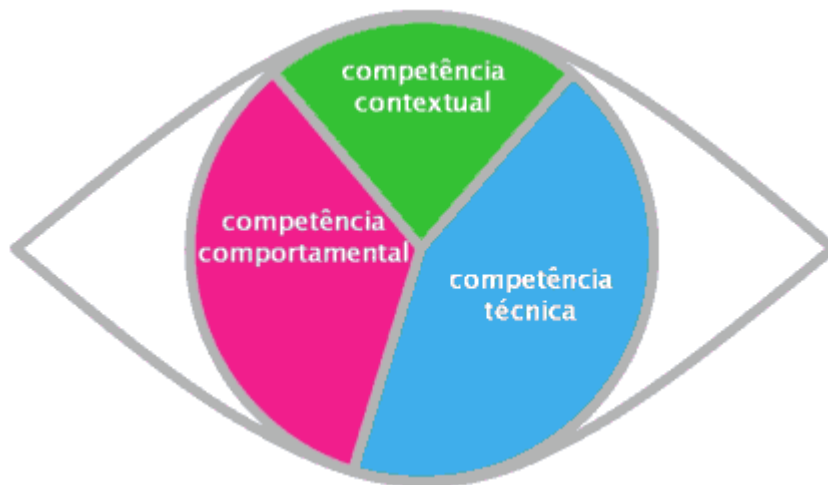
Além do padrão global para gerenciamento de projetos Guia PMBOK, o PMI publica padrões para o gerenciamento de programas — The Standard for Program Management — e de portfólios — The Standard for Portfolio Management.

Uma estrutura ou corpo organizacional frequentemente usado para o gerenciamento centralizado e coordenado dos projetos, programas e portfólios é o **escritório de projetos** (Project Management Office - **PMO**). As responsabilidades de um PMO podem variar desde fornecer apoio e suporte ao gerenciamento de projetos na organização até ser responsável pelo gerenciamento direto de um projeto, programa ou portfólio. [PMI 2008, p. 11] O PMO pode também consolidar informações e indicadores estratégicos dos projetos, programas e portfólios da organização.

Um PMO deve dar suporte aos gerentes de projetos, o que inclui: identificar e desenvolver metodologia, políticas, procedimentos, documentação, melhores práticas e padrões de gerenciamento de projetos; orientar com aconselhamento, treinamento e supervisão; gerenciar recursos compartilhados entre todos os projetos abrangidos pelo PMO; monitorar conformidade com políticas e padrões; coordenar a comunicação entre projetos.

Outros Referenciais Em Gerenciamento De Projetos

IPMA e ABGP



Uma entidade alternativa ao PMI é a Associação Internacional de Gerenciamento de Projetos — International Project Management Association (IPMA), mais difundida na Europa.

O documento IPMA Competence Baseline (ICB), produzido pela IPMA, é análogo ao PMBOK do PMI. O ICB, versão 3.0, 2006 (em inglês) pode ser baixado gratuitamente.

O "Olho da Competência" (ver figura) representa o olhar IPMA sobre as melhores práticas e competências dos Gerentes de Projetos. Simboliza o conjunto representado pelos 46 Elementos do ICBv3, em que 20 Elementos são da Competência Técnica, 15 da Competência Comportamental e 11 da Competência Contextual.

O modelo de certificação da IPMA é baseado em competências. O ICB é a base para o sistema IPMA 4 Level certification (4LC). Os quatro níveis de certificação profissional da IPMA são: D - Certified Project Management Associate; C - Certified Project Manager; B - Certified Senior Project Manager; A - Certified Projects Director. A certificação Nível D da IPMA é similar à CAPM do PMI, e a Nível C, à PMP.

A Associação Brasileira de Gerenciamento de Projetos (ABGP) está filiada, desde julho de 2002, à IPMA, se tornando IPMA Brasil. O ICB foi localizado e adaptado para o Brasil, produzindo o Referencial Brasileiro de Competências (RBC) [PDF] em Gerenciamento de Projetos, utilizado pela ABGP/IPMA Brasil na sua certificação de Gerentes de Projetos no Brasil.

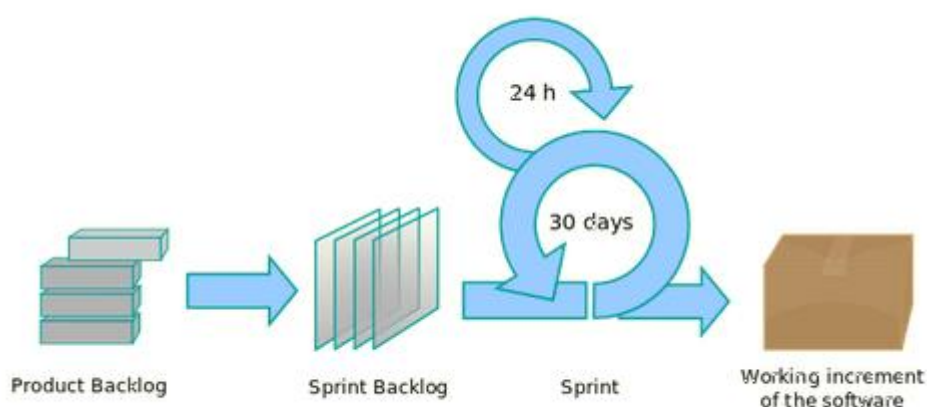
The Periodic Table of Project Management Competence Elements

IPMA PM Competence Element Groups										2.02 M Engagement & motivation	2.03 Sc Self-control
3.01 P Project orientation											
3.02 Pg Programme orientation	3.03 Pf Portfolio orientation							2.04 As Assertiveness	2.05 R Relaxation	2.06 O Openness	
3.04 Pp Project, program & portfolio implemen.	3.05 Po Permanent organization	1.01 Ps Project management success	1.02 Ip Interested parties	1.03 Rq Project requirements & objectives	1.04 Ri Risk & opportunities	1.05 Q Quality		2.07 Cy Creativity	2.08 Ro Results orientation	2.09 E Efficiency	
3.06 Bu Business	3.07 Sa Systems, products & technology	1.06 Po Project organization	1.07 T Teamwork	1.08 Pb Problem resolution	1.09 Ps Project structures	1.10 Sd Scope & deliverables		2.10 Co Consultation	2.11 Ne Negotiation	2.12 Cc Conflict & crisis	
3.08 Pe Personnel management	3.09 Hs Health, security, safety, & environment	1.11 TP Time & project phases	1.12 Re Resources	1.13 C Cost & finance	1.14 Cn Procurement & contract	1.15 Ch Changes		2.13 RI Reliability	2.14 Va Values appreciation		
3.10 Fi Finance	3.11 Le Legal	1.16 Cr Control & reports	1.17 In Information & documentation	1.18 Ca Communication	1.19 Su Project startup	1.20 Cs Project closeout		2.15 Et Ethics	Based on IPMA's ICB® www.ipma.ch		

Outro padrão de gerenciamento de projetos, originalmente desenvolvido pelo governo britânico e utilizado principalmente no Reino Unido, é o **PRINCE2 - Projects IN Controlled Environments**, atualmente mantido pela empresa AXELOS (a mesma que mantém o ITIL).

Métodos ágeis no gerenciamento de projetos

Várias técnicas e ferramentas baseadas em métodos ágeis tem sido aplicadas ao gerenciamento de projetos.



Scrum foi originalmente definida como "uma estratégia de desenvolvimento de produtos flexível e holística onde um time de desenvolvimento trabalha como uma unidade para atingir um objetivo comum", em 1986 pelos japoneses Hirotaka Takeuchi e Ikujiro Nonaka, no artigo "New Product Development Game". Tem sido difundida e utilizada como uma metodologia ágil para desenvolvimento de software iterativa e incremental, bem como um framework ágil para gerenciar o desenvolvimento de produtos e projetos. Preconiza princípios e técnicas como backlog de produto, reuniões "em pé" diárias, e ciclos curtos de desenvolvimento (duração de 2 a 4 semanas)

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.