ChatGPT

📊 Análise Crítica das Queries Dataview do Vault Empresarial

Visão Geral

Este documento analisa 16 queries **Dataview** utilizadas no vault empresarial, explicando a função de cada consulta e avaliando sua sintaxe, limitações e dependências (pastas, frontmatter YAML e funções do Dataview). Com base nessa análise crítica, são apresentadas sugestões de melhorias para otimizar as queries e recomendações de configuração do plugin Dataview, estrutura de vault e uso de frontmatter, alinhadas à prática real do vault empresarial.

Análise Detalhada das Queries Dataview

A seguir, cada query Dataview fornecida é descrita em detalhes, incluindo seu propósito e considerações sobre sintaxe e dependências:

Query 1: Total de Projetos

Função: Retorna o número total de projetos em andamento no vault. A query conta todos os arquivos de projeto localizados nas pastas `4-Projetos/Ativos` e `4-Projetos/Em-Desenvolvimento` (projetos ativos e em desenvolvimento) cujo frontmatter indica `type: "project"`. O resultado é exibido em uma tabela de uma linha com a métrica "Total de Projetos" e seu valor [1].

Sintaxe: Usa a função `length(rows)` para contar as páginas (notas) que atendem aos critérios. Observa-se o uso de múltiplas fontes no comando `FROM` através de `OR` para incluir duas pastas diferentes [1]. *Comentário:* A sintaxe ideal para múltiplas pastas seria `FROM "path1" OR "path2"` (não é necessário repetir o termo `FROM` antes do segundo caminho) [2]. No vault, a query está escrita como `FROM "content/4-Projetos/Ativos" OR FROM "content/4-Projetos/Em-Desenvolvimento"`, o que possivelmente ainda funciona, mas não é a forma mais limpa de expressar múltiplos diretórios.

Interdependências: Esta query depende fortemente da estrutura de pastas e do frontmatter: - Somente arquivos dentro das pastas especificadas serão contabilizados. Projetos fora dessas pastas (por exemplo, um projeto arquivado em outra pasta) **não serão incluídos**. - A condição `WHERE type = "project"` exige que cada nota de projeto tenha `type: project` no YAML. Se um projeto não tiver este campo ou houver um typo (`Type` maiúsculo, por exemplo), ele será ignorado. - A query não filtra por status além de presumir as pastas (e.g. não exclui projetos templates aqui, contando que estes estejam fora dessas pastas ou com outro status). No entanto, em outras queries nota-se a necessidade de filtrar `status != "template"` para evitar contar notas de template como projetos [3]. Portanto, garantir que projetos-modelo não estejam nestas pastas (ou tenham status apropriado) é essencial para a contagem correta.

Query 2: Projetos Ativos

Função: Conta quantos projetos estão *ativos no momento*, exibindo "Projetos Ativos" e o total [4]. Complementa a métrica anterior, focando apenas nos projetos em status ativo (exclui os em desenvolvimento).

Sintaxe: Semelhante à Query 1, mas aqui filtra `status = "active"` explicitamente além de `type = "project"` [5]. A origem de dados é apenas a pasta `4-Projetos/Ativos`, indicando que todos os projetos nessa pasta deveriam ter status "active".

Interdependências: - Depende do frontmatter `status: "active"` em cada projeto ativo. Se um projeto ativo não tiver o campo status preenchido corretamente, ou se um projeto em desenvolvimento estiver erroneamente nesta pasta, a métrica pode ficar inconsistente. - Depende da organização de que apenas projetos realmente ativos estejam na pasta *Ativos*. Essa separação pasta/status deve ser coerente: idealmente, **a pasta reflete o status**, reduzindo a necessidade de dupla filtragem. No entanto, aqui ambos são usados (pasta Ativos + status active), o que é redundante mas garante precisão caso haja alguma nota anômala na pasta. - Assim como antes, projetos templates são excluídos por estarem fora da pasta ou não terem status "active".

Query 3: Total de Pessoas

Função: Conta o total de pessoas (membros da equipe) ativas cadastradas no vault [6]. Retorna "Total de Pessoas" e o número de notas encontradas representando pessoas ativas.

Sintaxe: Busca em `content/2-Equipes` todas as notas cujo `type` é "person" e `status` é "active" [7]. Usa `length(rows)` para contar as entradas. Novamente apresentada em tabela de uma linha sem ID (ou seja, sem coluna de referência de arquivo).

Interdependências: - Exige que cada nota de funcionário/colaborador tenha `type: person` e `status: active` no YAML. Apenas essas serão contabilizadas. Membros inativos (status diferente) ou notas fora da pasta *2-Equipes* não entram na contagem. - Depende da convenção de que todas as pessoas estão armazenadas na pasta `2-Equipes`. Se algum perfil estiver em outro local ou se a equipe for subdividida em subpastas, a query (como escrita) incluirá subpastas automaticamente, mas **não incluirá** arquivos de pessoas fora de `2-Equipes`. - A sintaxe é direta. Poderia ser simplificada se, por exemplo, *todas* as pessoas ativas têm status active já – nesse caso, bastaria `WHERE type = "person" AND status = "active"` para qualquer local, mas limitar à pasta de pessoas evita considerar notas irrelevantes no vault.

Query 4: Capacidade Total

Função: Calcula a capacidade semanal total da equipe em horas, somando o campo `availability_weekly` de todas as pessoas ativas [8]. O resultado mostra "Capacidade Total" e, por exemplo, "320h/sem" (caso haja 8 pessoas de 40h/semana cada, por exemplo).

Sintaxe: Utiliza `sum(availability_weekly)` para somar o campo numérico de disponibilidade semanal de cada pessoa, concatenando `"h/sem"` para exibir a unidade [9]. Filtra pelo mesmo conjunto de pessoas (pasta 2-Equipes, type person, status active). Está em tabela sem ID, retornando uma única linha.

Interdependências: - Requer que todas as notas de pessoa tenham o campo `availability_weekly` preenchido com um **valor numérico** (horas disponíveis por semana). Se alguém não tiver esse campo ou se estiver como texto não-numérico (e.g. `"40 horas"` ao invés de `40`), o Dataview irá ignorar ou falhar no cálculo. A consistência da unidade (todas em horas/semana) é assumida e crucial. - Depende da definição de disponibilidade no YAML de cada pessoa. Alterações nesse campo atualizam automaticamente a soma. - Como a query soma sobre *todas* as pessoas ativas, ela fornece uma visão consolidada da capacidade total. Uma limitação é que não distingue diferentes departamentos ou tipos de contratação – apenas um total bruto. Caso seja necessário diferenciar (por exemplo, capacidade de time permanente vs temporários), seriam precisos filtros adicionais ou group by.

Limitação de sintaxe: A query em si é sintaticamente correta. Uma melhoria possível é especificar explicitamente `TABLE WITHOUT ID` (como foi feito) para não listar o identificador de cada nota somada, já que o objetivo é apenas o agregado.

Query 5: Utilização Média

Função: Calcula a **taxa média de utilização da equipe**, isto é, a porcentagem de horas alocadas em relação à capacidade total, em média, para todos os membros ativos [10]. Em outras palavras, indica quão ocupada, em média, a equipe está.

Sintaxe: Usa a fórmula `round((sum(allocated_hours) / sum(availability_weekly)) * 100, 1) + "%"`, ou seja, soma todas as horas alocadas e todas as disponibilidades semanais, divide e multiplica por 100, arredondando com uma casa decimal [11]. Filtra pelas notas de pessoas ativas como de costume. Resultado em tabela única sem ID, ex: "Utilização Média – 75%".

Interdependências: - Requer que cada pessoa tenha no YAML o campo `allocated_hours` (horas atualmente alocadas) além de `availability_weekly`. Ambos devem ser numéricos. Essa consulta envolve **dois campos** do frontmatter de cada nota de pessoa. - A precisão da métrica depende de todos os projetos estarem contabilizados nas horas alocadas de cada pessoa. Supõe-se que o campo `allocated_hours` em cada perfil é mantido atualizado somando todas as alocações daquela pessoa (talvez via uma query ou manualmente). Caso alguém tenha horas alocadas não atualizadas, a média resultante será incorreta. - A fórmula faz um cálculo de média ponderada global (soma total alocada / soma total disponibilidade), o que é equivalente a considerar a utilização agregada do grupo como um todo, não exatamente a média aritmética das utilizações individuais. **Observação:** Se quisesse a média simples das porcentagens individuais, teria que somar percentual de cada e dividir pelo número de pessoas. Aqui, porém, a métrica obtida representa a **utilização global** da capacidade disponível (o que é adequado para visão geral) [11]. - Limitação: se apenas uma parte da equipe tem alocação (por exemplo, novos funcionários com 0h alocadas), a média global diminui proporcionalmente. Pode ser interessante, em futuros refinamentos, calcular também mediana de utilização ou distribuir por equipes.

Query 6: Budget Total

Função: Soma o orçamento (`budget`) de todos os projetos em andamento (ativos ou em desenvolvimento) [12]. Fornece uma visão do investimento total nos projetos atuais, exibindo "Budget Total" e o valor em reais (prefixado com "R$").

Sintaxe: Semelhante às primeiras queries de projetos, faz `sum(budget)` nos arquivos das duas pastas de projetos, concatenando `"R$ "` para formatar como monetário [12]. Filtra `type =`

`"project"`. Não exclui nenhum status exceto *template* implícito pela pasta (mas **ver comentário abaixo**).

Interdependências: - Cada nota de projeto precisa ter um campo `budget` numérico (presumivelmente o orçamento aprovado). Projetos sem budget preenchido seriam considerados como 0 no somatório (Dataview normalmente trata ausência como null e ignora na soma, efetivamente 0). - O uso de duas pastas com OR indica que **projetos concluídos ou arquivados não estão sendo somados**, apenas os ativos e em desenvolvimento. Isso faz sentido se o interesse é no montante investido em projetos correntes. Caso queira o total histórico de orçamento de todos os projetos, seria necessário incluir outras pastas ou não filtrar por status/pasta. - A query como escrita não exclui explicitamente projetos templated. Se houver um arquivo de template de projeto nessas pastas com um campo budget (por exemplo, um valor de exemplo), ele seria somado indevidamente. Seria prudente filtrar `WHERE type="project" AND status != "template"` para garantir exclusão de templates, similar a outras queries [3]. No contexto do vault, possivelmente os templates estão fora destas pastas ou marcados de forma a não serem contados. - Dependência de frontmatter: a unidade monetária não está guardada no valor (guarda-se apenas número), e a formatação "R$ " é adicionada na query. Essa prática é boa, separando dado de apresentação. Contudo, note que somar moedas assume que todos budgets estão na mesma moeda (BRL), o que parece ser o caso.

Query 7: ROI Projetado

Função: Calcula o retorno financeiro projetado total dos projetos, multiplicando cada budget por 3,47 (347%) e somando tudo [13]. A escolha do fator 3,47 sugere que se espera, em média, um retorno ~3,47 vezes o investimento (talvez uma métrica histórica da empresa). O resultado mostra "ROI Projetado" e o valor em reais.

Sintaxe: Usa `sum(budget * 3.47)` para calcular a soma dos retornos esperados projeto a projeto [14]. Assim como as métricas anteriores, filtra projetos nas pastas de Ativos e Em-Desenvolvimento com `type = "project"`.

Interdependências: - Depende diretamente do campo `budget` de cada projeto (numérico) e de supor um multiplicador fixo (3,47) aplicado uniformemente. Esse multiplicador está **hardcoded** na query. Se a expectativa de ROI mudar, é necessário atualizar a fórmula manualmente em todas as queries relevantes (aqui e no insight de ROI médio, por exemplo). - Assim como em *Budget Total*, só considera projetos atuais (nas duas pastas). Projetos futuros ou passados fora dessas pastas ficam de fora. - Como esse ROI é calculado uniformemente, pode não distinguir projetos com diferentes potenciais de retorno. Em cenários mais complexos, talvez o ROI projetado devesse ser um campo específico de cada projeto (ou variando por tipo de projeto), mas no contexto dado a fórmula global é aplicada para simplificar. - *Limitação e melhoria:* caso a empresa queira detalhar ROI por projeto, seria interessante listar cada projeto com seu ROI esperado (há até uma query listando ROI por projeto individual mais adiante). A soma total, isolada, dá visão agregada mas não mostra distribuição. Ainda assim, a métrica é útil para ter ideia do retorno global esperado.

Query 8: Lista de Projetos com Dados Reais

Função: Gera uma lista-tabular de todos os projetos em andamento (ativos e em desenvolvimento), exibindo várias colunas-chave de cada projeto: Status, Projeto (nome com link), Owner (responsável), Budget, Esforço (em semanas) e Prioridade [15]. É essencialmente um *dashboard* de projetos detalhado.

Sintaxe: A query compõe uma tabela com colunas personalizadas: - `status` (texto do status do projeto), - `link(file.link, title)` para exibir o nome do arquivo como link clicável [16], - `owner` (campo do responsável no YAML, que parece ser um wikilink para a pessoa), - `budget` formatado como `"R$ " + budget`, - `effort_weeks` concatenado com "w" (semanas de esforço previstas), - `priority` (prioridade do projeto).

Filtra `FROM "4-Projetos/Ativos" OR "4-Projetos/Em-Desenvolvimento"` com `WHERE type = "project"` [17]. Ordena por `priority DESC, budget DESC` (prioriza projetos de prioridade maior e, dentro da mesma prioridade, orçamento maior primeiro).

Interdependências: - Esta query depende de *diversos campos no frontmatter de projetos*: `status`, `owner`, `budget`, `effort_weeks`, `priority`. Se algum não estiver presente em uma nota, a coluna sairá vazia para aquele projeto. - Depende também da consistência desses campos: - `owner` parece ser armazenado como um wikilink (`[[Nome da Pessoa]]`). Isso é bom, pois exibe como link e integra com a nota da pessoa. Contudo, note que na coluna aparece o texto do link (provavelmente o nome da pessoa). Se quisesse exibir só o primeiro nome, ou uma propriedade da pessoa, seria necessário algo mais complexo (ex: dataview JS ou usar campos relacionais). - `status` deve ser preenchido e refletir se está ativo ou em desenvolvimento. Curiosamente, a query não filtrou por status específico (apenas pela pasta), então ela listará todos projetos dessas pastas independente do valor exato de status (presumivelmente, na pasta Ativos todos têm status "active" e na Em-Desenvolvimento status "em desenvolvimento" ou similar). - `priority` deve ser um valor que permita comparação para sort (ex: "high", "medium", "low" ou numérico). Se for texto ("alta", "média, "baixa"), o sort pode estar ordenando alfabeticamente, o que nem sempre corresponde à ordem lógica. Caso as prioridades sejam palavras, uma melhoria seria mapear para números ou usar `choice()` para forçar a ordem, ou definir prioridades como números internamente. - *Sintaxe:* Esta query poderia ter usado `TABLE WITHOUT ID` porque estamos definindo colunas personalizadas e não queremos a coluna implícita do arquivo. Se não foi usado, o Dataview por padrão incluiria uma coluna inicial com o link da página. **Como incluímos manualmente o link do projeto**, seria recomendável usar `TABLE WITHOUT ID` para evitar duplicidade (a não ser que Dataview suprima o ID automaticamente quando se usa `file.link` - mas pela documentação, o `WITHOUT ID` é necessário [18]). Provavelmente a intenção era não exibir a coluna default; então acrescentar `WITHOUT ID` aqui seria uma melhoria de sintaxe. - Por fim, essa lista é sensível à quantidade de projetos; se o número crescer muito, pode ser útil paginar ou agrupar por status. Atualmente, todos projetos ativos e em desenvolvimento são mostrados em uma única tabela longa (ordenada por prioridade e budget).

Query 9: Capacidade da Equipe

Função: Lista todos os membros da equipe ativos, apresentando informações de capacidade e uso individual: Nome, Cargo, Departamento, horas disponíveis por semana, horas alocadas e a porcentagem de utilização de cada pessoa [19]. É um panorama da carga de trabalho por pessoa.

Sintaxe: A query monta uma tabela com colunas: - `name` (nome do membro, do YAML), - `role` (cargo/função), - `department`, - `availability_weekly` com "h/sem" concatenado, - `allocated_hours` com "h" concatenado, - a porcentagem calculada por `(allocated_hours / availability_weekly) * 100` arredondada [20].

Origem de dados: pasta `2-Equipes`, `WHERE type = "person" AND status = "active"` (todas pessoas ativas). Ordena por `allocated_hours DESC` para ver primeiro quem tem mais horas alocadas.

Interdependências: - Requer que cada nota de pessoa tenha os campos: `name`, `role`, `department`, `availability_weekly`, `allocated_hours`. Se algum faltar, sua coluna sairá vazia (ex: se alguém não tiver `department`, ficará em branco). - A fórmula de utilização individual `allocated_hours / availability_weekly` tem o pressuposto que ambos são números. Se `availability_weekly` de alguém for 0 ou estiver ausente, isso seria problemático (divisão por zero ou valor indefinido). Portanto, garante-se que todos ativos têm disponibilidade > 0. - A classificação por horas alocadas descendentemente ajuda a identificar rapidamente quem está mais carregado. Contudo, não considera o percentual; uma pessoa com 30h alocadas de 30h disponíveis (100% util) apareceria abaixo de alguém com 32h de 40h (80% util), pois 32 > 30 apesar do percentual menor. Se o interesse for ordenar por percentuais de utilização, poderia ordenar por `(allocated_hours/availability_weekly)` em vez de horas absolutas. Depende do foco desejado (carga absoluta vs relativa). - Assim como na Query 8, aqui também seria conveniente `TABLE WITHOUT ID` para não ter coluna de referência (a menos que se queria rapidamente abrir a nota da pessoa – mas o próprio nome podendo ser wikilink seria melhor). Atualmente, a query não usa `WITHOUT ID`, então é provável que a primeira coluna seja o link do arquivo da pessoa. Porém, também exibimos o `name` que provavelmente repete o nome do arquivo. Seria mais elegante fazer o nome como link: `link(file.link, name)` para unir as duas coisas, ou usar `WITHOUT ID` e exibir apenas o nome. **Sugestão:** Adotar `TABLE WITHOUT ID name AS "Nome", ...` ou `link(file.link, name) AS "Nome"` para evitar redundância. - Depende da estrutura de que todos membros da equipe residem em `content/2-Equipes`. Isso está ok conforme padrão do vault.

Query 10: Projetos por Status (Métricas Consolidadas)

Função: Agrupa os projetos por status e conta quantos projetos há em cada status, mostrando também alguma métrica agregada (no caso desta query, apenas a quantidade) [21]. Permite visualizar a distribuição de projetos pelos diferentes estados (ex: quantos em Ideia, Planejamento, Em Progresso, Concluídos, etc., dependendo dos status existentes).

Sintaxe: Utiliza `GROUP BY status` após filtrar projetos nas pastas Ativos/Em-Desenvolvimento [22]. As colunas definidas são `status` e `length(rows)` como "Quantidade". Isso produz uma linha por cada valor distinto de status encontrado, com a contagem de projetos naquele status. A cláusula `SORT length(rows) DESC` ordena os status do mais frequente para o menos frequente.

Interdependências: - Apoia-se no campo `status` preenchido em cada projeto. Se houver inconsistências (ex: um status escrito diferente ou maiúsculas diferentes), o Dataview tratará como status distintos. Por exemplo, "Em andamento" vs "em andamento" seriam duas linhas separadas. **Padronização do campo status é fundamental** para que o agrupamento seja correto. - Limita as fontes às pastas Ativos e Em-Desenvolvimento, então status como "Concluído" ou "Cancelado" (se existirem em outra pasta) não apareceriam. Ou seja, esta métrica está focada nos projetos atuais (correntes) agrupados pelos seus status atuais. Se quiser uma visão de todos os projetos do vault por status (incluindo concluidos), teria que incluir outras pastas ou retirar a restrição de pasta, adicionando filtro `status != "template"` apenas. - A query assume que `type = "project"` em todos, garantindo que está contando apenas notas de projeto (se por acaso na pasta houver notas de outro tipo, seriam excluídas). - *Sintaxe:* Como discutido, faltou `WITHOUT ID` aqui. Ao usar `GROUP BY`, o Dataview por padrão costuma não listar o ID, mas sim apresentar as categorias agrupadas como se fossem cabeçalhos. No entanto, para consistência e para evitar qualquer coluna de link de arquivo, `TABLE WITHOUT ID ... GROUP BY status` seria apropriado [18]. Provavelmente a saída atual já não mostra coluna de arquivo (Dataview usa o campo group como primeiro col agrupador, não um link), então está aceitável. - Essa métrica é útil para ver concentração de projetos em certos status (e.g. se muitos estão em “ Em Progresso” vs poucos em “ Em Teste”). Talvez no vault os status sejam indicados

por emojis (há uso de `choice(status, ...)` em outras queries para ícones [23]). Aqui, entretanto, o output provavelmente mostra o texto do status. Uma melhoria cosmética poderia ser usar essa mesma função `choice()` ou símbolos para ficar visual (por exemplo, trocar "active" por um ícone). Mas isso é secundário.

Query 11: Projetos por Owner (Responsável)

Função: Agrupa os projetos por responsável (owner) e conta quantos projetos cada pessoa lidera [24]. Essa query dá visibilidade da distribuição de projetos entre os gestores, identificando concentração de responsabilidade.

Sintaxe: Similar à anterior, mas com `GROUP BY owner` e colunas `owner` (como "Responsável") e `length(rows)` ("Projetos") [25]. Ordena pelo número de projetos decrescente.

Interdependências: - Depende do campo `owner` em cada nota de projeto. Pelos dados, `owner` é provavelmente preenchido com um wikilink para a pessoa (ex: `owner: [[Pedro Vitor]]`). Ao agrupar, Dataview pode mostrar o resultado possivelmente como link (ou texto do link). Deve-se ter atenção: se alguns projetos tiverem o nome do responsável escrito como texto simples e outros como `[[wikilink]]`, o Dataview pode tratar como valores diferentes (um caso seria "Pedro Vitor" vs "[[Pedro Vitor]]"). É importante padronizar: **sempre usar wikilink** ou sempre texto, para evitar entradas duplicadas. Dado que na query Insight há `owner = "[[Pedro Vitor]]"` [26], entende-se que usam wikilink. Provavelmente Dataview ao agrupar mostre o display text do link (o nome). - Projetos sem owner definido (campo vazio) seriam agrupados em uma categoria nula. Idealmente todos os projetos têm um responsável atribuído. - Assim como Query 10, está limitada às pastas de projetos atuais. Portanto, mostra responsáveis *dos projetos em andamento*. Se alguém tem projetos concluídos (fora dessas pastas), não conta aqui. Também, se um responsável não tiver nenhum projeto atual, ele não aparece. - *Sintaxe:* Poderia igualmente usar `TABLE WITHOUT ID`. Atualmente agrupar por owner provavelmente suprime o ID por padrão, mas não custa padronizar. - Uma possível melhoria de insight seria mostrar também o total de budget por owner, para ver carga financeira por responsável (mas isso não foi feito aqui; mantém-se simples com contagem). Do jeito que está já alerta se alguém (como Pedro Vitor) tem muitos projetos.

Query 12: Distribuição de Budget

Função: Lista o budget de cada projeto, ordenados do maior para o menor [27]. Serve para visualizar a distribuição individual de orçamento por projeto – essencialmente uma lista classificando projetos pelo valor de investimento.

Sintaxe: Define colunas `"R$ " + budget` como "Budget por Projeto". A única outra informação exibida é implícita: por padrão, sem `WITHOUT ID` aqui, provavelmente o Dataview incluirá a coluna do File (link para o projeto) ou, se não, possivelmente o file.name foi pretendido. Mas na Query 12 vemos apenas a linha 83980-83983, parece que só uma coluna foi definida ("Budget por Projeto") [27] sem indicar o nome do projeto na própria query. Se `WITHOUT ID` não foi usado (e não foi), o Dataview deve listar o *arquivo* como primeira coluna, o que aqui funcionaria: a coluna implícita "File" mostrará o nome do projeto (linkado), e ao lado a coluna "Budget por Projeto" com o valor. Assim cada linha é "<Projeto> – R$ X". Ou seja, usou-se um truque: não listar explicitamente o nome do projeto para deixar o ID padrão suprir isso. - Filtra `FROM` as duas pastas de projetos, `WHERE type = "project"`, e `SORT budget DESC` para ordenar do maior budget ao menor [28].

Interdependências: - Depende do campo `budget` em todos projetos (se algum não tem, ele aparecerá provavelmente com blank ou como 0). - Como supõe as mesmas pastas de projetos em andamento, esta distribuição não mostra projetos fora do pipeline atual. - *Sintaxe e melhorias:* Essa query está quase duplicada pela próxima (Projetos por Faixa de Budget). Aqui listamos *todos* os projetos com seu budget, ordenados. Seria útil talvez exibir também o nome do projeto na própria query para clareza, em vez de depender da coluna implícita. A próxima query (Faixa de Budget) tenta talvez algo diferente, vamos comentar lá. Mas uma melhoria seria combinar Query 12 e 13 em uma só que realmente mostre faixas. - Se a intenção era visualizar a dispersão de budgets, talvez um gráfico seria mais apropriado. Como em Obsidian não há gráfico nativo, listar ordenado é ok. Para fins de análise, a tabela resultante permite notar se há 1 ou 2 projetos consumindo a maior parte do budget (pelo valor) e muitos pequenos, etc.

Query 13: Projetos por Faixa de Budget

Função: A intenção aparente é mostrar projetos categorizados por faixa de orçamento. Contudo, do jeito que está implementada [29], a query lista novamente os projetos ordenados pelo budget, porém exibe o budget e o nome do projeto em colunas invertidas (Budget e Projeto). Provavelmente houve a ideia de criar faixas (low, medium, high budget) mas não foi concluída na sintaxe.

Sintaxe: Define colunas `"R$ " + budget AS "Budget"` e `file.name AS "Projeto"` [30], e ordena por budget desc. Em essência, isso produz uma tabela de duas colunas: Budget (valor) e Projeto (nome do arquivo do projeto). Não há `GROUP BY` nem filtro específico de faixa, então cada linha é um projeto. A diferença em relação à Query 12: - Aqui não se confia na coluna implícita do arquivo, pois explicitamente coloca `file.name` como Projeto. - Entretanto, `file.name` dá o nome do arquivo *sem link*. Portanto, esta tabela mostra nome do projeto como texto simples, não clicável, e o budget ao lado. Enquanto a anterior possivelmente mostrava link. - Filtragem igual: projetos nas duas pastas, sem exclusão de templates.

Interdependências e Observações: - Parece redundante com Query 12. Possivelmente, a ideia era agrupar budgets em faixas (ex: >50k, 10k-50k, <10k), mas isso exigiria usar uma expressão condicional ou múltiplas queries. Como não foi implementado, Query 13 acaba servindo pouco além de duplicar a lista de budgets. - Para realmente ver *faixas*, uma sugestão seria: - Usar `GROUP BY case(when budget >= 50000 then "Alto", when budget >=10000 then "Médio", else "Baixo")` ou algo semelhante. Dataview não tem um `CASE` fácil, mas poderia usar múltiplos `WHERE` ou o plugin DataviewJS. - Ou simplesmente apresentar os top N projetos por budget (já tem uma query de "Projetos de Alto Valor" como alerta). - Assim, do jeito atual, Query 13 não acrescenta muita informação nova. Poderia ser removida ou reformulada. - Em termos de sintaxe, ela demonstra o uso de `file.name` para obter o nome do arquivo sem link. Isso pode ser útil em cenários onde não se quer link, mas aqui possivelmente não era necessário desligar o link. - Interdependência: depende do nome do arquivo estar legível. Se os arquivos de projeto tiverem identificadores em vez de nomes claros, melhor usar frontmatter `title`. A consulta poderia usar `file.link` ou `link(file.path, title)` para mostrar um título customizado. No vault, talvez o nome de arquivo seja o próprio nome do projeto, então `file.name` é adequado.

Query 14: Horas Alocadas (Resumo de Capacidade)

Função: Soma o total de horas alocadas atualmente em todos os membros da equipe [31]. Complementa a "Capacidade Total" (Query 4) indicando quantas horas da capacidade total já estão comprometidas.

Sintaxe: Muito semelhante à Query 4, mas sum sobre `allocated_hours` em vez de disponibilidade, adicionando "h" como sufixo [31]. Filtra as notas de pessoas ativas (pasta 2-Equipes, type person, status active).

Interdependências: - Requer o campo `allocated_hours` preenchido numericamente para todos os membros ativos. Se alguém não tiver valor (ex: novo funcionário ainda sem alocação, mas não colocou 0), é possível que Dataview trate como 0 implicitamente ao somar (ignorando null). É prudente padronizar: pessoas sem alocação explícita poderiam ter `allocated_hours: 0` para clareza. - Depende da atualização manual ou automatizada desse campo conforme os projetos vão consumindo horas. **Limitação:** O vault parece não ter um vínculo automático entre projetos e horas das pessoas, então é provável que a atualização de `allocated_hours` seja manual ou via algum script. Uma eventual automação poderia recalcular isso somando tarefas/projetos atribuídos. - Ao lado da "Capacidade Total", este valor permite calcular as horas livres (próxima query) e a taxa de utilização global (Query 5, já calculada diretamente). Não há problemas de sintaxe aqui; é uma simples soma.

Query 15: Horas Livres (Resumo de Capacidade)

Função: Calcula o total de horas livres por semana na equipe, ou seja, capacidade total menos horas alocadas [32]. Indica quanta folga a equipe ainda tem na agenda coletiva.

Sintaxe: Usa a expressão `(sum(availability_weekly) - sum(allocated_hours)) + "h"` [32] para subtrair as horas alocadas da capacidade total, adicionando "h" no fim. Filtragem igual às demais de capacidade (pessoas ativas).

Interdependências: - Depende das duas somas anteriores (conceitualmente). Na prática, repete o cálculo dentro da query, portanto depende dos mesmos campos que Query 4 e 14. Se um desses campos faltasse para alguma pessoa, o cálculo seria afetado. - A interpretação desta métrica é direta, mas assume que `allocated_hours` não excede `availability_weekly` para ninguém (se alguém estiver sobrealocado além de suas horas semanais, essa conta ainda considerará horas livres possivelmente negativas ou diminuirá o total livre). No geral, espera-se allocated <= availability por pessoa, então o total livre faz sentido. - *Sintaxe:* Está correta e simples. Vale notar que essas três queries (Capacidade Total, Horas Alocadas, Horas Livres) poderiam ser condensadas em uma única query com múltiplas colunas, já que todas percorrem o mesmo conjunto de notas. Por exemplo, uma tabela com colunas "Capacidade Total", "Alocadas", "Livres" poderia ser gerada numa única passada. Entretanto, separá-las permite formatar cada uma possivelmente como um *card* separado no dashboard e estilizar individualmente. - **Dependência de estrutura:** Como antes, garantir que a pasta e filtros pegam todos os membros desejados (nenhum ativo ficou fora).

Query 16: Utilização Média (Resumo de Capacidade)

Função: Repete a métrica de utilização média da equipe (já vista na Query 5) dentro da seção de capacidade [33]. Provavelmente para ter o número de % de forma isolada como um card no dashboard de capacidade.

Sintaxe: Mesma fórmula de Query 5: `round((sum(allocated_hours) / sum(availability_weekly)) * 100, 1) + "%"` sobre pessoas ativas [34]. Tabela sem ID, uma linha.

Interdependências: - Idênticas às da Query 5. A duplicação aqui indica interdependência também na manutenção: se um cálculo tivesse que mudar, precisaríamos atualizar em dois lugares. Sempre que

possível, evitar duplicações reduz erro; contudo, às vezes duplicar permite exibir a métrica em seções diferentes do dashboard sem dependência de outra query. - Nenhuma novidade adicional além do já discutido na Query 5. Apenas reforça a dependência de frontmatter consistente.

(As 16 queries analisadas acima correspondem às principais consultas fornecidas. A seguir, faremos um diagnóstico geral e sugestões de melhorias, englobando também outras queries auxiliares de "Por Departamento", "Alertas" e "Insights", que aparecem no vault como complementares.)

(Extra) Query: Capacidade por Departamento

Função: Agrupa as pessoas por departamento para sumarizar capacidade, alocação e utilização por departamento [35]. Mostra quantas pessoas em cada departamento, total de horas disponíveis, total de horas alocadas e % de utilização média daquele departamento.

Sintaxe: Usa `GROUP BY department` nas pessoas ativas, e colunas agregadas: `length(rows)` (conta pessoas), soma de availability, soma de allocated, e percentual calculado (soma alocadas / soma disponíveis *100) [36]. Ordena pelo total de horas alocadas desc.

Interdependências: - Depende do campo `department` preenchido para cada pessoa. Qualquer pessoa sem departamento ficará agrupada como valor vazio (ou "Indefinido"), poluindo os dados – então deve-se padronizar que todo membro tem um departamento no YAML. - Oferece uma visão muito útil para ver qual departamento está mais sobrecarregado (percentual mais alto ou próximo dos 100%) ou subutilizado. No entanto, note que a coluna "Utilização" calculada aqui é também uma média ponderada por horas do departamento (não a média simples das pessoas daquele dept). - *Limitações:* Se houver poucos indivíduos em um departamento, percentuais extremos são menos significativos, mas a visualização ainda é válida. - *Sintaxe:* Correta. Como antes, `WITHOUT ID` poderia ser usado para suprimir coluna de arquivo (aqui provavelmente Dataview não coloca, já que está agrupado). - Interpretação: esta query evidencia dependência entre **metadados pessoais** (departamento) e **funções do Dataview** (group by, sum). É um bom exemplo de como o frontmatter estruturado (campo department em cada pessoa) permite análises de alto nível.

(Extra) Queries de Alertas de Capacidade

Função: Identificar indivíduos sobrecarregados e subutilizados. Há duas queries: - Pessoas **Sobrecarregadas**: lista membros com utilização > 80% [37]. - Pessoas **Subutilizadas**: lista membros com utilização < 30% [38].

Cada tabela mostra Nome, Cargo e % de utilização, ordenando decrescentemente (sobrecarregados) ou crescentemente (subutilizados) para destacar quem está no topo de cada categoria.

Sintaxe: As duas queries filtram a partir de `content/2-Equipes` onde `(allocated_hours / availability_weekly) > 0.8` (ou `< 0.3`) além de `type = "person" AND status = "active"` [39] [40]. As colunas são calculadas como percentual semelhante às anteriores.

Interdependências: - Dependem novamente dos campos de horas de disponibilidade e alocadas por pessoa. Se algum estiver faltando, a pessoa não será listada (ou possivelmente nem considerada, já que a condição não pode ser avaliada). - A escolha dos limiares (80% e 30%) é baseada em critérios de negócio para alerta. São fixos na query. Se a empresa quiser ajustar esses thresholds, é fácil alterar os valores na condição. - A presença dessas queries significa que a empresa monitora **equilíbrio de carga**. Isso traz implicações metodológicas: precisa-se definir ações (como de fato há Recomendações

imediatas pedindo redistribuição de carga [41]). - *Observação:* Uma potencial melhoria de sintaxe seria incluir o nome do departamento ou equipe da pessoa para contexto, mas talvez não necessário se se conhece cada nome. - Essas queries ilustram bem a interdependência entre **dados de pessoas** e **regras de negócio codificadas** nas queries. Caso a carga horária de alguém mude, eles entrarão ou sairão automaticamente desses resultados.

(Extra) Query: Projetos de Alto Valor

Função: Lista projetos cujo orçamento é alto (≥ R$ 50.000) [42]. É um filtro de projetos por valor, para destacar aqueles de maior investimento.

Sintaxe: Filtra nos projetos das pastas Ativos/Em-Desenvolvimento com condição `budget >= 50000` [43]. Colunas: Projeto (link com título), Budget (formatado), Owner, Status. Ordena por budget descendente.

Interdependências: - Depende do campo `budget` preenchido e comparável numericamente. Notas de projeto sem budget ou com budget menor que 50000 serão excluídas. - Depende também do threshold fixo de 50k. Se esse valor for considerado "alto valor" hoje, pode mudar no futuro (inflação, mudança de escala dos projetos). Nesse caso, ter esse número centralizado ou parametrizado seria útil, mas Dataview não permite variável global facilmente; então isso teria que ser ajustado manualmente. - Os campos exibidos (owner, status) precisam estar preenchidos nos projetos; senão aparecerão vazios. - *Sintaxe:* Correta. Talvez faltou `TABLE WITHOUT ID` se não desejarem a coluna file default (porém aqui a primeira coluna é já um link formatado, então provavelmente não há coluna extra). - Essa query cruza dados de projetos (budget, status, owner) para fins de *insight*, mostrando dependência de frontmatter consistente nos projetos de maior porte.

(Extra) Queries de Insights Baseados em Dados Reais

No vault há três pequenas queries de insights: 1. **Concentração de Projetos:** verifica quantos projetos uma pessoa específica (Pedro Vitor) gerencia [44]. 2. **Capacidade Disponível:** calcula novamente as horas livres por semana (mesma lógica da Query 15) mas apresentando em frase [45]. 3. **ROI Médio:** calcula a porcentagem média de ROI (347%) e receita projetada total [46].

Sintaxe: São queries em tabela `WITHOUT ID` com duas colunas "Insight" e "Detalhe", cada uma retornando uma linha com um pequeno texto: - *Concentração:* `'Pedro Vitor gerencia ' + length(rows) + ' projetos'` como detalhe, filtrado por `owner = [[Pedro Vitor]]` [26]. - *Capacidade Disponível:* similar à Horas Livres, resultando em frase do tipo "Xh livres por semana" [45]. - *ROI Médio:* texto fixo "347% com receita projetada de R$ Y" onde Y = sum(budget*3.47) [46].

Interdependências e Observações: - O insight de concentração está *hardcoded* para Pedro Vitor. Isso expõe uma dependência direta do conteúdo da query a um dado do vault (o nome do gestor principal). Se futuramente outra pessoa passar a ter mais projetos, a query não se autoatualiza para essa outra pessoa – continuaria mostrando Pedro Vitor. Portanto, está mais para uma *frase de destaque fixa* do que uma consulta genérica. Uma melhoria seria automatizar esse insight para pegar quem tem mais projetos (por exemplo, usando a Query 11 e extraindo o top1). Atualmente, depende do valor fixo e do link `[[Pedro Vitor]]` existir. - O insight de capacidade livre é exatamente a Query 15 reformatada em frase. Depende das mesmas coisas (campos de horas das pessoas) e não acrescenta nova dependência. - O insight de ROI médio assume que todos projetos têm ROI 3.47x, resultando sempre em 347%. Ele calcula sum(budget*3.47) mas depois converte esse número em receita projetada. Para obter 347%, provavelmente fizeram `(sum(budget*3.47)/sum(budget) * 100)` internamente ou, mais

*simples, sabem que a média será 3.47x, então colocaram fixo "347%". No código vemos que 347% está escrito literalmente no texto, e sum(budget*3.47) fornece a receita total esperada [46]. Isso significa que se todos projetos têm ROI 3.47, o ROI médio é invariavelmente 347%. Portanto, essa query está meio *codificando* um resultado que não muda a menos que mude o fator. - Em termos de sintaxe, todas usam `TABLE WITHOUT ID` e concatenam strings, o que demonstra o poder do Dataview de criar frases dinâmicas. Também demonstra limite: não há forma fácil de incluir condicionais complexas nas strings sem partir para DataviewJS. - Dependência: o insight de ROI médio depende do fator no texto estar alinhado com o fator do cálculo. Se alguém mudar o fator 3.47 no cálculo mas esquecer de alterar o "347%" no texto, ficaria inconsistente. Isso exige atenção manual.

Após a análise individual das queries acima, podemos extrair padrões de uso e pontos de atenção gerais. A próxima seção resume um diagnóstico geral da sintaxe e das limitações observadas, bem como as interdependências entre estrutura do vault, frontmatter e funcionamento do Dataview.

Diagnóstico Geral: Sintaxe, Limitações e Interdependências

Organização em Pastas vs. Metadados: As queries evidenciam uma forte dependência tanto da **estrutura de pastas** quanto dos **campos YAML**. Em vários casos, há redundância – por exemplo, filtrar por `WHERE type = "project"` e ao mesmo tempo restringir a pastas específicas de projetos. Isso sugere que o vault categoriza notas por pastas e também etiqueta por tipo no frontmatter. Essa dupla organização pode ser positiva (belts and suspenders) para evitar erros, mas também indica trabalho manual extra para manter consistência. Idealmente, **cada conceito deveria ser definido de uma forma primária**: - Ou a pasta já define o tipo (ex.: tudo em `4-Projetos` é type=project), - Ou o frontmatter define e as queries poderiam varrer o vault inteiro confiando nos campos (`WHERE type="project"` sem especificar pasta).

No vault empresarial, optou-se por ambos. É importante notar que arquivos fora das pastas esperadas serão ignorados pelas queries, mesmo que tenham o frontmatter correto. Portanto, **a correta localização das notas** é essencial para o sistema funcionar. A interdependência entre pastas e frontmatter é alta: se alguém mover um arquivo de projeto ativo para fora de `4-Projetos/Ativos`, ele some do dashboard; se esquecer de marcar `status: active`, ele também some.

Sintaxe do Dataview e Padrões de Código: Em geral, as queries usam a sintaxe declarativa do Dataview adequadamente (comandos TABLE, WHERE, GROUP BY, etc.). Alguns pontos sintáticos a aprimorar: - Uso de `OR` **no FROM**: Conforme mencionado, escrever `FROM "A" OR "B"` é válido e suficiente [2]. Nas queries foi escrito `OR FROM` repetidamente, o que não é necessário. Padronizar para `FROM "path1" OR "path2" OR "path3"` melhora a legibilidade. - **Coluna de ID implícito**: Várias queries esqueceram ou optaram por não usar `TABLE WITHOUT ID`. Como resultado, podem estar mostrando uma coluna extra com o nome do arquivo. Nas listas detalhadas (projetos, equipe), isso causa duplicação de informação (ex.: nome da pessoa aparecendo duas vezes). A recomendação é usar `TABLE WITHOUT ID` sempre que a primeira coluna não for intencionalmente o identificador da página. Isso dá mais controle sobre o que é exibido [18]. - **Formatação e concatenação**: Observou-se concatenação de strings para adicionar unidades ("h", "%", "R$"). Isso é correto. Poderia-se usar funções de formatação (por ex. `dateformat` para datas, ou até `currencyformat` que o Dataview introduziu [47]). No entanto, os métodos manuais adotados funcionam bem. Só cuidado para manter consistência (por ex., ora usam `"h"` ora `"h/sem"`, mas isso faz sentido já que em alguns contextos era semanal). - **Uso da função** `choice()`: No vault há exemplos de `choice(status, " ","◇"," ",...)` para substituir status por emojis [23]. Essa função, na documentação, está definida para sintaxe booleana básica (if-else) [48]. No entanto, parece que o Dataview aceita uma extensão onde `choice(field,`

`opt1, opt2, ...)` escolhe com base em índices numéricos ou valores fixos. É algo não bem documentado e pode confundir. Uma abordagem mais clara seria usar `emoji::` no frontmatter ou mapear via um `CASE` no DataviewJS. O uso de `choice()` dessa forma funciona, mas é potencialmente frágil se o plugin mudar. Diagnóstico: sintaxe não padrão, mas funcional – só tenha em mente a possibilidade de precisar ajustar caso o Dataview evolua. - **Agrupamentos e agregações:** As queries mostram o Dataview lidando bem com `GROUP BY` e funções agregadas (`sum`, `length`). Uma limitação a notar: não é possível calcular porcentagens dentro do mesmo GROUP BY facilmente para cada grupo (exceto via DataviewJS). Por isso, a query Por Departamento calculou a utilização por departamento manualmente (soma alocado / soma disponibilidade do grupo). Está correto. Em outros casos, faltou talvez alguma métrica agregada extra (como somar budgets por status ou owner – isso poderia ser feito adicionando outra coluna `sum(budget)` junto com `length(rows)`). - **Limitações intrínsecas do Dataview:** As consultas são *estáticas* no sentido que rodam ao abrir a nota ou mandar atualizar. Se dados mudam (ex: alguém altera um frontmatter em outro arquivo), é preciso reatualizar o painel. O vault provavelmente lida com isso através do hábito de usar `Ctrl+R` (atalho mencionado) para recarregar métricas [49]. Isso é esperado, não um defeito – apenas parte do funcionamento (Dataview não autoatualiza em tempo real a menos que o arquivo com a query seja reaberto ou o comando refresh seja acionado). - **Dependência de valores fixos em queries:** Alguns critérios estão hardcoded (limites de 80%, 30%, budget 50000, ROI factor 3.47, nome "Pedro Vitor"). Isso mostra onde o sistema não é *declarativo a partir de dados*, mas sim *codificado na lógica*. Tais dependências precisam ser gerenciadas: se a pessoa "Pedro Vitor" sair da empresa ou se quiserem mudar o threshold de alerta, é necessário editar a query manualmente. O plugin não suporta facilmente referenciar um valor de outra nota ou uma configuração global. Uma prática para contornar isso pode ser: criar uma nota de configuração (ex: `Settings` com campos globais) e usar DataviewJS para ler de lá. Porém, isso adiciona complexidade. No estado atual, deve-se pelo menos documentar esses parâmetros fixos para fácil localização.

Limitações de Dados e Fronteiras do Sistema: Pelas queries podemos inferir algumas limitações do próprio sistema empresarial: - O acompanhamento de horas alocadas por pessoa parece ser manual (campo `allocated_hours`). Não há uma query somando tarefas ou projetos atribuídos para gerar isso automaticamente. Isso requer disciplina da equipe em atualizar sempre que um projeto é atribuído ou removido de alguém. - Não há query para verificar **inconsistências** (por ex., um projeto sem owner, ou somatório de horas alocadas dos projetos de uma pessoa divergindo do campo no perfil dela). Tais validações poderiam ser feitas com DataviewJS ou ao menos consultas de "dados faltando". Sua ausência indica que esse controle talvez seja externo ou simplesmente não foi implementado ainda. É uma área a considerar (ver sugestões). - O Dataview, por si, **não modifica dados**, apenas lê. Se há necessidade de ações automáticas (como marcar alguém como sobrecarregado e notificar), isso precisa de integração externa (script, Plugin Templater, etc.). As queries de alerta fornecem a informação, mas não tomam ação. - **Desempenho e escala:** As queries atuais percorrem essencialmente duas pastas principais (projetos e equipes). Isso é pequeno em termos de escopo. Se o vault crescer muito (milhares de notas de projeto, histórico longo), algumas queries (especialmente se expandidas para buscar em todo vault) podem ficar lentas. O Dataview indexa dados e costuma ser eficiente, mas agrupar e somar muitos itens pode ter custo. No presente, não parece um problema, mas é bom monitorar. Dividir dados (por exemplo, arquivar projetos antigos fora do escopo das queries ativas) é uma forma de manter performance – e pelo visto isso já é feito via segregação de pastas.

Interdependência Vault ↔ Query (Resumo): Em suma, as queries não são isoladas – elas codificam suposições sobre: - Onde os arquivos estão (pastas), - Como estão anotados (frontmatter), - Quais valores consideram relevantes (ex.: status != "template", ou excluir "Templates" no path [26]), - Quais campos existem e em que formato (strings vs números, links vs texto).

Essa “contrato” entre os dados e as consultas precisa ser mantido. Qualquer mudança estrutural (renomear uma pasta, mudar um nome de campo YAML) exigirá atualizar as queries correspondentes. O diagnóstico geral é que o sistema está bem estruturado, porém bastante **rigoroso**: exige disciplina para não quebrar as premissas, e duplicações de lógica (como ROI factor, thresholds) que precisam de governança.

Sugestões de Melhoria nas Queries

Com base nos pontos levantados, seguem sugestões para tornar as queries mais eficientes, claras e fáceis de manter:

- **1. Unificar fontes por tipo em vez de pastas múltiplas:** Sempre que possível, use apenas `WHERE type = ...` em combinação com exclusões mais genéricas, em vez de listar várias pastas com OR. Por exemplo, para "Total de Projetos", poderia-se usar `FROM "4-Projetos"` (toda a pasta de projetos) com `WHERE type="project" AND status != "template"`. Isso incluiria *todos os projetos*, independente de subpasta (Ativos, Em Desenv, Concluídos, etc.), excluindo apenas templates. Caso se deseje focar só nos projetos não concluídos, pode filtrar por `status` específico. A vantagem é que adicionando uma nova subpasta (ex: "Concluídos"), não precisaremos atualizar todas as queries – apenas o filtro de status se for o caso. **Nota:** Ao fazer isso, é preciso garantir que as notas de projeto carreguem `type: project` e `status` adequadamente em qualquer pasta.

- **2. Remover redundâncias pasta vs status:** Como complemento do item acima, se a estrutura de pastas já separa status, podemos confiar nela e simplificar o filtro. Ou vice-versa, se confiarmos no frontmatter status, a pasta pode ser mais livre. Atualmente há duplicidade (pasta Ativos + status active). Seria mais elegante escolher um como fonte da verdade. Uma melhoria poderia ser:

- *Ou* reorganizar para ter uma pasta única `4-Projetos` com subpastas ou tags para status, usando apenas `status` no filtro (ex.: `WHERE status="Active"`),
- *Ou* manter subpastas e remover o campo `status` do YAML (embora o campo também sirva para exibir nos relatórios, então talvez não).
- Ou pelo menos, no código da query, não filtrar duas vezes. Exemplo: Query 2 poderia ser apenas `FROM "4-Projetos/Ativos" WHERE type="project"` (já presumindo que tudo ali é ativo).

Isso reduz a chance de conflitos (como alguém botar status "active" numa nota que está na pasta Em-Desenvolvimento por engano, o que query 2 contaria indevidamente).

- **3. Padronizar uso de `WITHOUT ID` e colunas de identificação:** Revisar todas as queries de tabela e adicionar `WITHOUT ID` onde aplicável para evitar colunas de arquivo duplicadas [18]. Em especial:
- Na lista de projetos (Query 8), usar `TABLE WITHOUT ID` já que definimos nossa coluna Projeto linkada.
- Na lista de equipe (Query 9), ou usar `WITHOUT ID` ou transformar `name` em link: `link(file.link, name) AS "Nome"`. Assim, uma única coluna serve de identificação e acesso à nota.
- Nas tabelas agrupadas (Queries 10, 11, Por Departamento), adicionar `WITHOUT ID` por consistência, garantindo que só mostre as colunas definidas.

Isso deixa as tabelas mais limpas e sob controle do conteúdo que escolhemos mostrar.

- **4. Consolidar queries de métricas repetidas:** Notamos que "Capacidade Total", "Horas Alocadas", "Horas Livres", "Utilização Média" foram realizadas como queries separadas (no Resumo de Capacidade) [50] [31] [32] [33]. Embora isso facilite a exibição isolada de cada valor, também significa 4 passes sobre os mesmos dados. Uma alternativa eficiente:
- Fazer *uma única query* DataviewJS que calcula todas essas métricas de uma vez e exibe, por exemplo, em uma lista ou tabela customizada. DataviewJS permitiria usar JavaScript para calcular e formatar múltiplos outputs com uma única varredura do dv.pages().
- Ou, se quiser ficar só em DQL, uma `TABLE` com múltiplas colunas e uma só linha: ex:

```
TABLE without id
  sum(availability_weekly) + "h" AS "Capacidade Total",
  sum(allocated_hours) + "h" AS "Horas Alocadas",
  (sum(availability_weekly) - sum(allocated_hours)) + "h" AS "Horas
Livres",
  round(sum(allocated_hours) / sum(availability_weekly) * 100,1) + "%"
AS "Utilização Média"
FROM "content/2-Equipes"
WHERE type="person" AND status="active"
```

Isso retornaria exatamente uma linha com as 4 colunas desejadas. Poderia então ser estilizada via CSS para aparecer em formato de cards, por exemplo.

Consolidar reduziria redundância e risco de alguma dessas queries ficar incoerente com outra se houver mudanças futuras. No entanto, pode-se argumentar que separá-las permite cache do Dataview e atualização independente. De qualquer forma, do ponto de vista de *manutenção*, é mais fácil ajustar uma query unificada do que quatro separadas.

- **5. Automatizar highlights dinâmicos:** Em vez de fixar no código o nome "Pedro Vitor" para concentração de projetos, considerar uma abordagem dinâmica. Por exemplo, poderíamos criar um DataviewJS snippet:

```
let projectsByOwner = dv.pages('"4-Projetos"')
                       .where(p => p.type === "project" && p.status !=
"template")
                       .groupBy(p => p.owner);
if(projectsByOwner.length) {
  let topOwner = projectsByOwner.reduce((a,b) => (b.rows.length >
a.rows.length ? b : a));
  dv.paragraph(`**Concentração de Projetos:** ${topOwner.key} gerencia $
{topOwner.rows.length} projetos.`);
}
```

Isso encontraria quem tem mais projetos e mostraria. Essa lógica poderia substituir o insight fixo. É mais complexa que a atual query simples, mas elimina a necessidade de mudar manualmente o nome do gestor caso mude.

Se não quiser usar JS, outra ideia: adicionar no YAML de pessoas um campo computado externamente (ex: cada pessoa tem `projects_count`) e atualizar isso via script periodicamente. Mas isso começa a

complicar demais. Para uma doc interna, sugerimos pelo menos anotar que "Pedro Vitor" está hardcoded e deve ser alterado se necessário.

- **6. Introduzir queries de** validação de dados**: para reforçar a governança, usar Dataview para encontrar itens que violem padrões:
- Exemplo: notas de projeto *sem owner* – Query: `LIST FROM "4-Projetos" WHERE type="project" AND !owner` (lista projetos onde owner não está definido). Isso ajudaria a manter ninguém sem responsável.
- Projetos sem budget: `WHERE type="project" AND budget = 0` (se interpretarmos budget 0 ou null como falta de orçamento).
- Pessoas sem alocação definida (talvez ok deixar 0, mas se quiser listar quem está totalmente livre).
- Pessoas sem departamento definido: `WHERE type="person" AND !department`.

Essas queries de "QA" poderiam ficar numa seção de administração (talvez não no dashboard executivo visível a todos, mas em alguma nota de manutenção do sistema). Isso reduz erros silenciosos.

- **7. Comentários e documentação no código das queries:** O Dataview permite inserir comentários dentro do bloco usando `%% comment %%`. Seria benéfico documentar dentro das próprias queries complexas qualquer lógica não óbvia. Por exemplo, na query ROI Médio Insight, anotar `%% multiplica budgets por 3.47 para obter receita projetada total e 347% fixo como ROI médio %%`. Isso ajuda futuros mantenedores a entender por que foi feito assim.

- **8. Tornar critérios configuráveis centralmente (quando possível):** Embora o Dataview não permita facilmente referenciar valores externos, uma estratégia simples:

- Criar uma nota (ex: `Configurações Dashboard`) listando em YAML algo como:

```
high_load_threshold: 0.8
low_load_threshold: 0.3
high_budget_threshold: 50000
roi_factor: 3.47
```

O Dataview DQL não consegue usar esses valores diretamente, mas DataviewJS sim (poderia buscar a nota e ler dv.current().high_load_threshold etc. se a query estiver embutida nessa nota). Ou até Templater plugin poderia inserir esses valores.

Se não for para automatizar, ao menos ter essa nota de documentação ajuda a lembrar de atualizar todos os lugares relevantes quando um valor mudar (um passo manual, mas guiado).

Em resumo, a melhoria aqui é mais de **governança** do que técnica: isolar e documentar parâmetros para fácil ajuste.

- **9. Utilizar propriedades *derivadas* e calculadas:** O Dataview suporta inline fields (como `Status:: Ativo` dentro da nota) e também implicitamente sabe de algumas coisas (por ex.: `file.mtime` última modificação, etc.). Podemos pensar em usar essas propriedades:

- Exibir, por exemplo, na lista de projetos, a última atualização de cada projeto, usando `file.mtime` ou um campo `updated` do YAML. Isso daria insight de qual projeto não é atualizado há muito (podendo indicar estagnação). Se relevante, adicionar colunas com essas infos seria trivial.

- Aproveitar tags: se projetos tivessem tags como `#prioridade/alta`, poderia filtrar ou estilizar. No entanto, o vault parece preferir campos estruturados a tags, o que é válido.

- **10. Verificar a possibilidade de** Views compostas **ou** DataviewJS **para coisas complexas:** Alguns relatórios poderiam ser combinados:

- Ex: Um *dashboard de compliance* poderia mesclar vários pequenos insights (como os "Tendências e Recomendações" fazem). Já que Obsidian suporta seções em markdown, poderia formatar insights lado a lado com um pouco de HTML/CSS. O vault já inclui HTML e até referencia partials nunjucks para o Publish [51]. Isso indica que para publicação estática, talvez as queries são substituídas por algo.

  Porém, internamente no Obsidian, se quisesse, DataviewJS pode criar elementos DOM personalizados. Por exemplo, colorir em vermelho percentuais acima de 80%. Esse tipo de melhoria visual pode ser conseguido via DataviewJS scripts ou CSS classes.

- **Exemplo de DataviewJS simples:** ordenar e pegar top N. Em Dataview DQL às vezes falta como fazer LIMIT (embora haja `LIMIT` em tasks ou em alguns contexts e de fato aparece `LIMIT 5` em outra parte do vault [52]). Confirmando, Dataview DQL suporta `LIMIT` para limitar resultados. Poderíamos usar em queries se quiséssemos mostrar só top 5 projetos de alto budget (mas no caso já fizeram alerta separado).

Em suma, **centralizar o que puder, simplificar filtros redundantes, padronizar sintaxe e tornar parâmetros explícitos** são as linhas gerais das sugestões. Implementá-las reduzirá a possibilidade de erro humano (por exemplo, esquecimento de atualizar uma query enquanto outra foi atualizada) e facilitará evolução do vault (como adicionar novos status, novos departamentos, etc., sem ter que modificar muitas consultas).

Configuração do Plugin e Estrutura do Vault: Recomendações

Por fim, apresentamos recomendações voltadas à configuração do plugin Dataview, à organização do vault e às convenções de frontmatter, de acordo com as práticas do sistema empresarial:

- **Configurações do Dataview Plugin:** Certifique-se de habilitar as opções necessárias no plugin:
- **Refresh Automático:** O Dataview possui uma opção de auto-refresh em modo Preview (ou um comando para refresh). Garantir que a equipe saiba usar (via o atalho `Ctrl+R` que foi documentado [49]) é importante para sempre ver dados atualizados. Pode-se também configurar *Dataview Refresh on file change* se disponível, para atualizar as queries quando notas relacionadas mudarem (versões recentes do Dataview/Datacore podem suportar isso).
- **JavaScript Queries (DataviewJS):** Habilitar se não estiver ativo, já que em alguns casos avançados usamos `dataviewjs`. No vault já existe uso de `dataviewjs` (vimos referências no arquivo compilado [53]). Então é algo já em uso – manter ativado.
- **Max Result Limit:** Se houver configurações de limite de resultados padrão, avaliar se precisam ajuste. Em vaults grandes, talvez aumentar limites ou permitir paginação. Por enquanto não parece crítico.

- **Persistência de Cache:** Existe uma configuração experimental para persistir o cache do Dataview entre sessões (evita reindexar tudo a cada abertura do vault). Em vaults empresariais maiores, habilitar isso melhora a performance no startup. Verifique se a versão do plugin tem essa opção (algumas discussões do GitHub mencionam *persist cache* [54]).

- **Estrutura de Vault e Pastas:** A estrutura atual com prefixos numéricos (0-Dashboard, 2-Equipes, 4-Projetos, etc.) é boa para ordenar e distinguir seções. Algumas sugestões:

- Mantenha **pasta de Templates separada** e fora das pastas de conteúdo ativo. Parece que há algo como "Templates" referenciado [55] . Idealmente, todos templates residem em, por exemplo, `Templates/` ou `_templates/` e as queries então podem facilmente excluir essa pasta inteira (`AND !contains(file.path, "Templates")` como foi feito). Isso é preferível a marcar cada template com status=template, embora o vault faz ambos. Poderia padronizar: *ou* mover templates para fora de `4-Projetos` (se atualmente estão dentro), *ou* ao menos garantir todos têm status=template e então usar só essa condição.
- Utilize subpastas **consistentes para status**: por exemplo, se futuramente houver `4-Projetos/Concluídos`, já planejar as queries para ou incluir ou excluir conforme necessário. Uma convenção pode ser: queries de dashboard operacional excluem "Concluídos" (projetos encerrados), enquanto relatórios históricos poderiam incluir.
- Crie uma pasta ou seção para **Metadados e Configurações** (se seguir a sugestão de ter nota de parâmetros ou documentos técnicos). Talvez um `9-Administrativo` para notas sobre o sistema (documentação, changelog do vault, configurações).

- **Nome dos arquivos**: manter nomes de arquivo amigáveis e únicos, pois eles acabam aparecendo nas queries (e no publicado no Netlify, via permalink talvez). No YAML do Dashboard integrado vemos `permalink: /dashboard-executivo/` [56] e provavelmente projetos podem ter permalinks ou usar o file.name para URL. Garantir que nomes de projeto não colidam e sejam identificáveis (já deve ser prática, só reforçando).

- **Boas Práticas de Frontmatter YAML:** O vault já utiliza extensivamente o frontmatter, o que é ótimo. Recomendações para mantê-lo organizado:

- **Definição de Esquemas:** Documente o "esquema" de cada tipo de nota (projeto, pessoa, dashboard, etc.). Por exemplo:

    - Projeto: campos obrigatórios (title, owner, budget, status, priority, market_segment, etc.).
    - Pessoa: name, role, department, availability_weekly, allocated_hours, status, etc.
    - Dashboard: title, type, status, etc (como no Dashboard Central YAML tem layout, priority, owner, etc. que servem ao Eleventy).

  Essa documentação pode viver numa nota ou em um README. Isso ajuda na padronização. Inclusive, se possível, use o plugin **Metadata Menu** ou **Templater** para garantir a criação de notas novas já com o frontmatter correto. Talvez vocês já tenham templates para isso (ex: um template de novo projeto). Automatizar a aplicação de templates evita que alguém esqueça um campo.

- **Valores Consistentes e Tipos:**

    - Use sempre o mesmo formato para campos similares. Ex: `status` aparentemente é sempre minúsculo e em inglês ("active", "template") – manter assim uniformemente (ou

se mudar para português, mudar em todos). Idem para `department` (usar nomes consistentes, talvez até predefinidos), `role` (cargos – definir nomenclatura padrão), etc.
    - Campos numéricos: nunca incluir unidade no YAML (e.g. não por "40h", e sim `40` no YAML e adicionar "h" na exibição). Isso já está sendo feito, continue assim.
    - Campos de porcentagem calculada: não armazenar porcentagem estática no YAML, calcule quando necessário. Vi que não armazenam (utilização sempre calculada). Correto.
    - Links para notas: quando um campo referencia outra nota (ex: owner -> pessoa; ou em outros contextos, projeto poderia linkar departamento ou vice-versa), usar o formato `[[Nome da Nota]]`. Isso cria vínculos no grafo e permite o Dataview mostrar como link. O vault já faz isso para owner e possivelmente poderia fazer, por exemplo, se cada projeto tivesse um campo de membros envolvidos (lista de pessoas) – mas não vi esse campo, talvez não necessário.

- **Ordem e legibilidade:** Manter o frontmatter ordenado de maneira lógica (por ex., campos principais primeiro: title, type, status, depois campos específicos, depois técnicos como created, updated, etc.). Isso facilita edição manual e conferência. Novamente, templates ajudam nisso.

- **Campos técnicos de controle:** O YAML do Dashboard central tem `version`, `updated`, etc. [57] . Isso indica algum automation (talvez um script incrementa versão, ou updated timestamp). Mantenha esses campos atualizados via automação se possível (um script ao publicar ou um hook). Se não houver, considere implementar um método de ao menos logar mudanças importantes. Esses campos ajudam na governança (ex.: saber quando foi a última atualização de um dashboard ou nota).

- **Uso de Tags vs Campos:** No Obsidian, YAML tags (ex: `tags: [teste, exemplo]`) e inline tags (#tag) também contam como metadata. Pelo material, vocês não parecem usar tags, preferindo campos explícitos. Isso é bom para clareza. Apenas lembre que o Dataview pode filtrar por tags facilmente também. Se houver categorias transversais, poderia usar. Mas se não há necessidade, manter campos explícitos é mais estruturado.

- **Controle de Versão (Git) e Implantação:** Embora não diretamente parte do plugin Dataview, a infraestrutura de versão e publicação do vault é importante:

- Usar Git para versionar o vault (ou ao menos a parte de conteúdo) é altamente recomendado. Isso permite histórico de mudanças e recuperação se algo corromper. Dado o perfil empresarial, acredito que já adotam (talvez integrando com Netlify/Eleventy).
- Se vários editores mexem no vault, um fluxo Git com pull requests ajudaria a revisar alterações no YAML e nas queries para manter padrão (ex: code review de uma nova query Dataview).

- Netlify: Garantir que durante o processo de build (Eleventy) as queries Dataview sejam resolvidas ou substituídas. Notei no Dashboard Central publicado há placeholders `{{ metrics.activeProjects }}` etc. Isso indica que o conteúdo final do site é gerado via Eleventy scripts, não mostrando as queries do Dataview diretamente no site. Portanto, as métricas são provavelmente calculadas por um plugin Eleventy (ou até por um *data script* que parseia os markdowns). É importante manter essas duas fontes sincronizadas: as queries Dataview em Obsidian e a lógica de cálculo no site devem refletir o mesmo resultado. Seria interessante, se possível, reutilizar a mesma fonte de verdade – por exemplo, talvez extrair dados para um JSON e alimentar tanto Dataview quanto Eleventy. Mas isso foge do escopo aqui; apenas alertando que a **duplicação de lógica** (Obsidian vs Eleventy) pode ocorrer. Minimizar discrepâncias é chave (já que o Dashboard no Obsidian mostra dados reais e o site também deveria).

- **Treinamento da Equipe e Governança Contínua:** Como o material será usado para treinar a equipe:

- Assegure que todos entendam as convenções acima (pastas, frontmatter, como preencher campos).
- Defina um **"dono do sistema"** (um papel, possivelmente Pedro Vitor ou alguém de tecnologia) responsável por revisar periodicamente as queries e dados. Essa pessoa garantiria que, conforme o negócio muda (novos tipos de projetos, novas métricas desejadas), o sistema Obsidian/Dashboard seja atualizado apropriadamente.
- Incorpore feedback do time: se perceberem que alguma informação não está sendo capturada (por ex, necessidade de trackear prazo dos projetos – não vimos campo de deadline, talvez não precisem, mas se precisassem, seria um novo frontmatter a adicionar e query a criar).
- Documente dentro do vault (como estão fazendo) as funcionalidades para que novos membros possam se auto-orientar. Este documento técnico pode residir no vault mesmo, permitindo buscas. O Obsidian é bom para isso, pois o próprio documento vira parte do sistema de conhecimento da equipe.

Em resumo, o plugin Dataview está sendo utilizado de forma robusta para transformar o vault em um banco de dados vivo [58] [59]. Para aproveitar todo o potencial mantendo a confiabilidade: - **Mantenha os dados estruturados e consistentes** (via templates, revisão regular), - **Simplifique e centralize a lógica das queries** quando possível (evitando duplicações e tornando-as adaptáveis), - **Use os recursos do plugin e do Obsidian** (como dataviewJS, Metadata Menu, version control) para automatizar o que for viável, - **Monitore a saúde do sistema** com queries de auditoria e envolva a equipe na atualização constante.

Seguindo essas práticas, o sistema Obsidian Vault + Dataview do ambiente empresarial permanecerá **sustentável, confiável e escalável**, servindo tanto para o uso diário (dashboards, gestão) quanto para análise estratégica e publicação externa (via Netlify), com mínimo esforço manual de manutenção.

[Referências: Dataview plugin transformando vault em banco de dados [58]; Função do Dataview nas notas Obsidian [59]; Discussões de melhor uso do `TABLE WITHOUT ID` *[18] e do filtro por múltiplas pastas [2] foram consideradas na elaboração das sugestões acima.]*

Canvas de Integração do Sistema (Obsidian Vault Empresarial)

A seguir, apresenta-se um **Canvas ultra-complexo** em formato JSON (.canvas) compatível com Obsidian, modelando o sistema em 5 regiões (4 quadrantes + centro). O canvas contém mais de 60 nós conectando conceitos teóricos e práticos sobre o Obsidian, Dataview e a implementação empresarial discutida. Cada nó está anotado com conteúdo denso (múltiplas frases, listas e exemplos) e utiliza formatação Markdown e emojis para facilitar a visualização. Os quadrantes são:

- **Quadrante Superior Esquerdo – Fundamentos Teóricos:** conceitos base do Obsidian, estrutura de vaults e funcionamento do Dataview.
- **Quadrante Superior Direito – Aplicações Práticas:** usos concretos no contexto empresarial, como dashboards, automações e gestão de projetos/equipe.
- **Quadrante Inferior Esquerdo – Metodologias:** processos de construção do sistema, versionamento do vault e governança (padrões, templates, controles).

- **Quadrante Inferior Direito – Resultados e Métricas:** resultados obtidos e análises derivadas das queries Dataview (insights de projetos, capacidades, ROI etc.).
- **Centro – Integração do Sistema:** visão holística da arquitetura do vault, automações com agentes (scripts/bots), interação com LLMs (IA) e publicação externa (Netlify).

Os nós e conexões estratégicas refletem a hierarquia macro ↔ nano, garantindo coerência temática e fluidez. O JSON fornecido pode ser importado como um canvas no Obsidian para exploração interativa.


```
{
  "nodes": [
    {
      "id": "fundamentals_main",
      "type": "text",
      "x": 200,
      "y": 200,
      "width": 400,
      "height": 180,
      "color": "4",
      "text": "📚 **Fundamentos Teóricos**\nConceitos base do Obsidian e Dataview. Inclui a estrutura de **Vaults** (coleção de pastas/notas), uso de **metadados (YAML)** e princípios do plugin **Dataview**. Esses fundamentos permitem tratar notas como dados estruturados, transformando o vault em um **banco de dados vivo** [58]."
    },
    {
      "id": "obsidian_vault",
      "type": "text",
      "x": 50,
      "y": 400,
      "width": 380,
      "height": 160,
      "color": "4",
      "text": "🗄 **Obsidian & Vault**\nObsidian armazena notas em **Markdown** dentro de uma pasta raiz chamada *Vault*. O Vault empresarial é organizado hierarquicamente (por exemplo, `0-Dashboard`, `2-Equipes`, `4-Projetos`), o que facilita navegação. Todas as notas são arquivos locais - **controle total** do usuário sobre os dados. Um vault pode ser sincronizado via Git ou cloud, mantendo histórico de mudanças."
    },
    {
      "id": "yaml_metadata",
      "type": "text",
      "x": 50,
      "y": 600,
      "width": 380,
      "height": 180,
      "color": "4",
      "text": "🏷 **Frontmatter YAML**\nNo topo de cada nota, entre `---`, definimos metadados estruturados. Exemplo em uma nota de projeto:\n\n```yaml\ntitle: Projeto X\nowner: [[Pedro Vitor]]\nstatus: active\nbudget:
```

```
50000\n```\nEsses campos permitem consultas Dinâmicas. O **Dataview** lê esses dados para gerar listas e tabelas. A consistência no YAML (mesmos campos em notas do mesmo tipo) é essencial para resultados corretos."
    },
    {
      "id": "dataview_concept",
      "type": "text",
      "x": 50,
      "y": 820,
      "width": 380,
      "height": 150,
      "color": "4",
      "text": "📻 **Plugin Dataview**\nDataview transforma o vault em um pseudo-banco de dados, permitindo fazer **queries nas notas** [59]. Com Dataview, podemos listar notas que atendem a critérios, fazer somas, agrupamentos e exibir campos do frontmatter de forma dinâmica. Tudo via código em blocos ```dataview``` dentro das notas. É leitura/escrita *read-only* (não altera notas, apenas exibe consultas atualizadas automaticamente)."
    },
    {
      "id": "dql_basics",
      "type": "text",
      "x": 60,
      "y": 1000,
      "width": 360,
      "height": 170,
      "color": "4",
      "text": "💻 **Linguagem de Consulta Dataview (DQL)**\nAs queries Dataview usam sintaxe própria semelhante a SQL. Principais comandos:\n- `TABLE/LIST/TASK` – formato de saída (tabela, lista ou tarefas);\n- `FROM \"pasta\"` – de onde buscar notas (ou usar tags, ou todo vault);\n- `WHERE condição` – filtros usando campos (ex: `status = \"active\"`);\n- `GROUP BY campo` – agrega resultados por valor comum;\n- Funções: `sum()`, `length()`, etc., para cálculo.\nEx.: ```dataview\\nTABLE name, role FROM \"2-Equipes\" WHERE status=\"active\"```"
    },
    {
      "id": "folders_vs_fields",
      "type": "text",
      "x": 330,
      "y": 500,
      "width": 360,
      "height": 150,
      "color": "4",
      "text": "  **Pastas x Metadados**\nO sistema usa **pastas** para organizar por assunto (equipe, projetos) e **campos YAML** para detalhes (tipo, status, etc.). Pastas facilitam delimitar escopo (Dataview `FROM \"4-Projetos\"` pega tudo ali). Já os metadados permitem filtrar mais granularmente (`WHERE type=\"project\" AND status=\"active\"`). É importante que pastas e campos **concordem** – por ex., não deixar um projeto ativo fora
```

```
da pasta de ativos, nem errar o status no YAML. Essa redundância fornece segurança, mas requer disciplina de organização."
    },
    {
      "id": "data_consistency",
      "type": "text",
      "x": 330,
      "y": 700,
      "width": 360,
      "height": 160,
      "color": "4",
      "text": "✔ **Consistência dos Dados**\nPara as queries funcionarem, os dados precisam ser consistentes:\n- Mesmos campos para mesmo tipo de nota (todos projetos com `budget`, `owner` etc. preenchidos);\n- Valores padronizados (status pré-definidos para não ter variações como \"Em andamento\" vs \"em andamento\");\n- Tipos corretos (números como números – ex: `40`, não \"40h\" no YAML);\n- Atualização manual ou automatizada dos campos (ex.: manter `allocated_hours` de cada pessoa atualizado com as horas planejadas). Pequenas inconsistências podem levar a resultados incorretos ou notas ignoradas pelas queries."
    },
    {
      "id": "dataviewjs_note",
      "type": "text",
      "x": 340,
      "y": 880,
      "width": 340,
      "height": 140,
      "color": "4",
      "text": "⚙ **DataviewJS (Avançado)**\nAlém da sintaxe padrão (DQL), o plugin oferece DataviewJS – que permite escrever JavaScript para consultas mais complexas. Com `dataviewjs`, temos total flexibilidade: loops, condicionais, acesso às páginas via API. No sistema, DataviewJS pode ser usado para calcular algo não suportado nativamente (ex: identificar automaticamente quem gerencia mais projetos ou formatar resultados especiais). Requer mais conhecimento técnico, mas é um recurso poderoso para além das limitações do DQL."
    },
    {
      "id": "dv_limitations",
      "type": "text",
      "x": 330,
      "y": 1050,
      "width": 360,
      "height": 170,
      "color": "4",
      "text": "❗ **Limitações do Dataview**\nMesmo sendo muito útil, o Dataview tem limitações a considerar:\n- **Somente Leitura:** Não altera arquivos; automatizações que precisem mudar conteúdo requerem outras ferramentas.\n- **Atualização:** As queries refletem o estado do vault; porém, se o vault muda externamente, pode exigir comando de refresh no
```

```
Obsidian para atualizar a visualização.\n- **Complexidade de consultas:** Algumas coisas (ex: múltiplas condições complexas, junção de diferentes tipos de notas) podem precisar de DataviewJS ou preparação manual de dados.\n- **Performance:** Em vaults muito grandes, muitas queries podem deixar a interface lenta ao abrir a nota que as contém. É preciso testar escalabilidade e talvez limitar escopos ou usar `LIMIT`."
    },
    {
      "id": "applications_main",
      "type": "text",
      "x": 1300,
      "y": 200,
      "width": 420,
      "height": 150,
      "color": "2",
      "text": "🛠 **Aplicações Práticas (Sistema Empresarial)**\nComo esses fundamentos são aplicados no vault corporativo. A empresa utiliza Obsidian+Dataview como um **sistema de gestão integrada**, incluindo dashboards executivos, acompanhamento de projetos, gerenciamento de capacidade da equipe e automações de fluxo de trabalho. Nesta seção vemos exemplos concretos de uso do sistema para resolver necessidades do dia a dia empresarial."
    },
    {
      "id": "dashboard_exec",
      "type": "text",
      "x": 1150,
      "y": 380,
      "width": 380,
      "height": 180,
      "color": "2",
      "text": "📊 **Dashboard Executivo Central**\nUm painel central em Obsidian agregando todas as métricas-chave da empresa. Inclui indicadores como **total de projetos, capacidade utilizada, orçamento total e ROI projetado**. Gerado automaticamente via queries Dataview sobre notas reais [60] [12]. Por exemplo, mostra quantos projetos estão em andamento, quantas horas da equipe estão alocadas e quantas livres, etc., tudo em tempo real a partir dos dados no vault. Esse dashboard substitui planilhas manuais, fornecendo uma visão única para diretoria."
    },
    {
      "id": "project_pipeline",
      "type": "text",
      "x": 1150,
      "y": 580,
      "width": 400,
      "height": 170,
      "color": "2",
      "text": "  **Pipeline de Projetos**\nO sistema Obsidian acompanha o *portfólio de projetos* da empresa. Todos os projetos são notas em `4-Projetos/`, com status variando (ideia, em planejamento, em desenvolvimento,
```

```
finalizado). Uma query gera uma lista de projetos ativos e em desenvolvimento com seus status, responsáveis, esforço e prioridade [15]. Isso funciona como um *Kanban/portfolio*: dá para ver quais projetos estão em qual fase, quem é o responsável (campo *owner*) e priorização. Ajuda na alocação de atenção e recursos."
    },
    {
      "id": "team_capacity_dashboard",
      "type": "text",
      "x": 1150,
      "y": 760,
      "width": 400,
      "height": 180,
      "color": "2",
      "text": "  **Gestão de Capacidade da Equipe**\nAtravés de notas de *pessoas* e queries, o Obsidian se torna uma ferramenta de **gestão de equipe**. Exemplo: um relatório Dataview lista cada membro da equipe com seu cargo, disponibilidade e carga alocada [19]. Daí derivam-se indicadores como quem está sobrecarregado (>80% das horas) ou subutilizado (<30%) [37] [38]. Isso permite aos gestores reequilibrar tarefas. O dashboard de equipe substitui planilhas de capacity planning, mostrando sempre a informação atualizada conforme os projetos atribuídos são refletidos nos campos YAML das pessoas."
    },
    {
      "id": "financial_analysis",
      "type": "text",
      "x": 1150,
      "y": 960,
      "width": 410,
      "height": 170,
      "color": "2",
      "text": "💰 **Análise Financeira de Projetos**\nO sistema gera visões financeiras automaticamente. Exemplo: soma de todos os budgets dos projetos em andamento [12] dá o *Budget Total*. Também calcula o *ROI projetado* usando um fator padronizado (ex.: 3,47 vezes o investimento) [13]. Além disso, é possível listar projetos de maior orçamento (ex.: >50k) para destacar *projetos de alto valor* [42]. Essas análises, antes feitas manualmente, agora são instantâneas. Gestores podem ver se o portfólio está pesado em investimento e qual retorno esperar globalmente."
    },
    {
      "id": "alerts_system",
      "type": "text",
      "x": 1160,
      "y": 1160,
      "width": 390,
      "height": 140,
      "color": "2",
      "text": "⚠ **Alertas Automáticos**\nO vault possui queries que funcionam como um *sistema de alerta*. Exemplos:\n- Lista de pessoas com alocação acima de 80% (alerta de sobrecarga) [37].\n- Lista de pessoas com
```

```
alocação abaixo de 30% (alerta de subutilização) [38].\n- Projetos com budget muito alto (potencial risco/atenção) [42].\nEsses alertas são gerados automaticamente sem precisar procurar manualmente. Eles permitem agir preventivamente: redistribuir projetos, ajustar foco, etc. No vault, há até seções explícitas de \"  Alertas e Insights\" destacando esses pontos críticos."
    },
    {
      "id": "task_management",
      "type": "text",
      "x": 1540,
      "y": 400,
      "width": 360,
      "height": 150,
      "color": "2",
      "text":
"  **Gestão de Tarefas e Ações**\nAlém de projetos e pessoas, o sistema pode gerir tarefas (to-do's) diretamente no Obsidian. Usando caixas de seleção `- [ ]` e talvez o plugin *Obsidian Tasks* ou consultas Dataview, é possível listar tarefas pendentes por responsável, por projeto ou por prazo. No dashboard foram incluídas seções de **Próximas Ações** (esta semana, próximo mês) [61]. Essas tarefas estão no Markdown e podem ser agregadas. Por ex., `TASK FROM \"4-Projetos\" WHERE !completed` listaria todas tarefas não concluídas nos projetos, ajudando no follow-up. Assim, o vault serve também como ferramenta de acompanhamento de ações."
    },
    {
      "id": "popup_navigation",
      "type": "text",
      "x": 1550,
      "y": 580,
      "width": 350,
      "height": 150,
      "color": "2",
      "text": "🔍 **Navegação & Popups**\nA experiência no Obsidian é otimizada para facilidade de acesso às informações:\n- **Links e Pop-ups:** Todo nome de projeto ou pessoa está linkado à sua nota. Passar o mouse mostra um *preview popup* com detalhes [62], sem precisar abrir outra aba. Ex: ao ver um projeto listado, pode-se hover no responsável para ver seus dados de capacidade.\n- **Atalhos de Teclado:** A equipe utiliza atalhos do Obsidian (p.ex. `Ctrl+K` para busca de nota, `Ctrl+R` para atualizar queries [49]). Isso agiliza navegar entre partes do sistema.\n- **Backlinks/Grafo:** Cada nota sabe onde é referenciada. Por ex., uma nota de pessoa mostra todos projetos (notas) que a referenciam como owner, via backlinks. Isso funciona como CRM: clicando numa pessoa, vê-se seus projetos relacionados automaticamente."
    },
    {
      "id": "collab_sharing",
      "type": "text",
      "x": 1550,
      "y": 750,
```

```
      "width": 360,
      "height": 150,
      "color": "2",
      "text": "  **Colaboração e Transparência**\nUsando Obsidian como base, todo o time tem acesso a uma fonte única de informações. Em vez de múltiplas planilhas ou documentos separados, o Vault centraliza tudo, garantindo **uma versão da verdade**. Com controle de versão (Git) e sincronização, vários usuários podem contribuir (adicionar projetos, atualizar status) com histórico rastreável. Além disso, graças ao publish (Netlify), mesmo quem não usa Obsidian pode visualizar as informações em formato de site. Essa transparência melhora a comunicação interdepartamental – todos veem o mesmo dashboard e dados."
    },
    {
      "id": "automation_flows",
      "type": "text",
      "x": 1550,
      "y": 920,
      "width": 360,
      "height": 160,
      "color": "2",
      "text": "  **Automações de Fluxo de Trabalho**\nO sistema aproveita integrações e plugins para reduzir trabalho manual:\n- **Templates Automáticos:** criação de novas notas de projeto ou pessoa a partir de modelos predefinidos (via plugin Templater), já populando campos padrão.\n- **Atualização de Campos:** scripts podem somar automaticamente horas alocadas de projetos e atualizar o campo `allocated_hours` em cada nota de pessoa (por ex., um agente que percorre todos projetos e acumula horas por responsável).\n- **Notificações:** Em conjunção com ferramentas externas, as informações do vault podem gerar alertas no Slack/Email (ex.: um agente lê o JSON das queries e envia alerta se alguém > 100% alocado ou projeto atrasado). Embora não nativo do Obsidian, as *\"saídas\"* do vault podem alimentar automações."
    },
    {
      "id": "methodology_main",
      "type": "text",
      "x": 180,
      "y": 1480,
      "width": 420,
      "height": 150,
      "color": "6",
      "text": "  **Metodologia: Construção, Versionamento e Governança**\nEste quadrante aborda *como* o sistema Obsidian/Dataview foi construído e é mantido ao longo do tempo. Envolve práticas de engenharia de conhecimento: uso de templates, controle de versão (Git) do vault, governança dos dados (validação, revisão) e treinamento da equipe. Essas metodologias garantem que o sistema se mantenha confiável e evolutivo, evitando deterioração com o crescimento e mudanças."
    },
    {
      "id": "standard_templates",
```

```
      "type": "text",
      "x": 40,
      "y": 1650,
      "width": 380,
      "height": 170,
      "color": "6",
      "text": "🏷 **Templates Padronizados**\nPara garantir que novas notas sigam o padrão, utilizamos *templates* no Obsidian. Por exemplo:\n- Template de **Projeto**: estrutura YAML básica (campos de owner, budget, etc.), seções preformatadas (Objetivos, Status, etc.). Quando um projeto novo é iniciado, cria-se a nota a partir do template.\n- Template de **Pessoa**: campos padrão (nome, role, disponibilidade padrão, etc.).\n- Template de **Reunião** (se for o caso): campos de data, participantes, etc.\nEsses templates padronizam o conteúdo e reduzem erros de omissão de campos. No vault, provavelmente existe uma pasta `Templates/` com esses modelos. Assim, todo projeto entra já *no formato certo* para ser capturado pelas queries."
    },
    {
      "id": "folder_structure",
      "type": "text",
      "x": 40,
      "y": 1840,
      "width": 380,
      "height": 150,
      "color": "6",
      "text":
"🗂 **Estrutura de Pastas do Vault**\nO vault empresarial segue uma hierarquia bem definida:\n- `0-Dashboard-Executivo/` – contêm o dashboard central e possivelmente subnotas agregadoras.\n- `2-Equipes/` – notas de pessoas/equipe. Pode subdividir por departamento se desejado, ou simplesmente conter todas pessoas ativas.\n- `4-Projetos/` – notas de projetos. Dentro, subpastas por status (Ativos, Em-Desenvolvimento, Concluídos, Templates). Os queries focam nas pastas Ativos/Em-Desenv.\n- Outras pastas numeradas (1-..., 3-...) podem existir para outros conteúdos (ex.: documentação, conhecimento, etc.) não abordados aqui.\nEssa estrutura alinha com a necessidade de rápido filtro e mantém o vault organizado semanticamente."
    },
    {
      "id": "version_control",
      "type": "text",
      "x": 50,
      "y": 2000,
      "width": 380,
      "height": 160,
      "color": "6",
      "text": "  **Controle de Versão e Backup**\nO vault sendo um conjunto de arquivos Markdown, encaixa-se bem com **Git**. Recomenda-se manter o vault versionado:\n- Cada mudança (ex: atualização de status, adição de projeto) registrada num commit. Assim há histórico: *Quem mudou o quê e quando.*\n- Permite **reverter** em caso de erro (ex.: alguém apagou acidentalmente um campo YAML, podemos recuperar).\n- Facilita colaboração: via Git, múltiplos
```

```
usuários podem editar diferentes partes e mesclar.\nNo contexto atual, o vault possivelmente já é integrado a um repositório que também alimenta a publicação no Netlify. Além do Git, backups regulares (em nuvem interna ou Obsidian Sync) garantem resiliência do sistema contra perda de dados."
    },
    {
      "id": "data_quality_governance",
      "type": "text",
      "x": 430,
      "y": 1650,
      "width": 390,
      "height": 170,
      "color": "6",
      "text":
"  **Validação & Governança de Dados**\nPara manter a qualidade dos dados, implementamos processos de revisão:\n- **Queries de QA:** O vault pode ter notas com dataview listando incoerências, ex: projetos sem owner definido, campos numéricos faltando, pessoas sem projetos atribuídos (se esperado ter algum), etc. Essas notas atuam como checklist para regularização.\n- **Revisão Periódica:** Um responsável revisa mensalmente os metadados. Ex.: confere se todos os projetos ativos têm status correto ou se algum deveria ser arquivado. Confere se budgets estão atualizados.\n- **Atualizações Concomitantes:** Define-se procedimentos: quando um projeto muda de status para \"Concluído\", move a nota para pasta correspondente *e* atualiza seu status YAML. Treinar a equipe para fazer ambas as coisas evita desalinhamento. \n- **Documentação das Convenções:** Este próprio material serve de documentação. Deixar registrado no vault (nota de convenções) para novos usuários seguirem as regras."
    },
    {
      "id": "training_onboarding",
      "type": "text",
      "x": 430,
      "y": 1840,
      "width": 390,
      "height": 150,
      "color": "6",
      "text": "🎓 **Treinamento da Equipe**\nNem todos estão habituados a Obsidian e Dataview, então investe-se em onboarding:\n- Sessões de treinamento mostrando como navegar no vault, atualizar notas de projeto, interpretar os dashboards.\n- Guia rápido (cheat sheet) dos comandos e sintaxes principais caso precisem editar ou criar uma query (embora maioria das queries já esteja pronta, usuários avançados poderiam tentar novas consultas para análises ad-hoc).\n- Cultura de contribuição: encorajar times (PMs, líderes) a manterem seus dados atualizados no sistema. Por ex., quando um projeto muda fase, imediatamente refletir na nota. Quando um colaborador muda de disponibilidade, atualizar seu frontmatter.\n- Suporte contínuo: designar alguém (\"bibliotecário do vault\") para tirar dúvidas e garantir adesão às práticas."
    },
    {
```

```
      "id": "maintenance_evolution",
      "type": "text",
      "x": 430,
      "y": 2000,
      "width": 390,
      "height": 150,
      "color": "6",
      "text": "  **Manutenção e Evolução**\nO sistema não é estático – ele evolui com as necessidades da empresa:\n- **Atualização de Consultas:** Se novos campos forem adicionados (ex: campo *risco* em projetos), as queries e painéis devem ser ajustados para incorporá-los. Mantenha uma lista de onde modificar (por isso centralizar lógica é útil).\n- **Performance Tuning:** Se o número de notas crescer muito, talvez filtrar escopos ou dividir vault (por ano, por exemplo). Monitorar a responsividade.\n- **Novas Funcionalidades:** A empresa pode incorporar novos plugins (ex: Calendar, Maps) ou integrações (ex: integrar com Jira ou outros sistemas via export/import). O vault deve ser flexível para isso – mantendo a base de dados, mas permitindo complementos.\n- **Arquivamento:** Definir política de arquivar projetos antigos (mover para pasta `Arquivados` ou exportar para outro vault de histórico) para que o vault ativo fique enxuto e focado. As queries então precisam excluir esses arquivados."
    },
    {
      "id": "results_main",
      "type": "text",
      "x": 1320,
      "y": 1480,
      "width": 400,
      "height": 140,
      "color": "1",
      "text": "📈 **Resultados, Métricas e Insights**\nEste quadrante consolida os principais resultados obtidos a partir das queries Dataview. São as métricas e análises extraídas automaticamente do conteúdo do vault. Esses resultados alimentam decisões e mostram o valor do sistema. Inclui quantificações (números totais), distribuições (por status, por responsável, por departamento) e insights (alertas e indicadores calculados)."
    },
    {
      "id": "projects_overview",
      "type": "text",
      "x": 1170,
      "y": 1640,
      "width": 370,
      "height": 130,
      "color": "1",
      "text":
"  **Total de Projetos e Projetos Ativos**\nAtualmente, o vault indica:\n- **Total de Projetos em Andamento:** quantidade de projetos considerados no pipeline (soma de ativos + em desenvolvimento). Ex.: *${totalProjetos} projetos* 1 .\n- **Projetos Ativos:** subset que estão em execução no momento (status active) – *${projetosAtivos} projetos ativos* 63 .\nEsses números dão
```

```
ideia do portfólio corrente. Por ex., se Total=10 e Ativos=6, implicando 4 em
desenvolvimento (pré-execução). Tendências podem ser observadas ao longo do
tempo (ex: total crescendo indica entrada de novos projetos sem fechar
antigos)."
    },
    {
      "id": "team_overview",
      "type": "text",
      "x": 1170,
      "y": 1770,
      "width": 370,
      "height": 150,
      "color": "1",
      "text":
"  **Total de Pessoas e Utilização Média**\nMétricas globais da equipe:\n-
**Total de Pessoas Ativas:** Quantidade de colaboradores considerados no
cálculo (ex.: *${numPessoas} pessoas* ativas) [6] . Isso ajuda a contexto: as
demais métricas (capacidade total, etc.) referem-se a esse universo.\n-
**Utilização Média da Equipe:** Porcentagem média da capacidade utilizada [11]
– e.g. *75%*. Esse valor resume se, em média, a equipe está *sobrecarga* ou
*oca*. No exemplo, 75% significa que coletivamente 3/4 das horas disponíveis
estão alocadas em projetos. É uma média ponderada (leva em conta diferentes
cargas horárias). Caso a média suba demais (próximo de 100%), indica pouco
espaço para novos projetos sem realocar ou contratar."
    },
    {
      "id": "capacity_allocation_totals",
      "type": "text",
      "x": 1170,
      "y": 1920,
      "width": 380,
      "height": 130,
      "color": "1",
      "text": "⏱️ **Capacidade vs Alocação (Horas)**\nVisão macro semanal:\n-
**Capacidade Total:** Somatório de horas semanais disponíveis de toda equipe.
Ex.: *320h/sem* disponíveis no total (se 8 pessoas de 40h, p.ex.) [9] .\n-
**Horas Alocadas:** Somatório de horas já comprometidas com projetos. Ex.:
*240h* alocadas atualmente [31] .\n- **Horas Livres:** Diferença entre
capacidade e alocadas – *80h livres* (no exemplo) [32] .\nEsses números mostram
a folga existente: se horas livres = 0, significa todos estão 100% ocupados.
No exemplo, 80h livres sugerem possibilidade de alocar mais trabalho
(equivalente a 2 pessoas livres se 40h cada)."
    },
    {
      "id": "financial_totals",
      "type": "text",
      "x": 1170,
      "y": 2040,
      "width": 380,
      "height": 130,
      "color": "1",
```

```
      "text": "  **Budget Total e ROI Projetado**\nIndicadores financeiros agregados:\n- **Budget Total:** soma de orçamentos de todos projetos em andamento. Ex.: *R$ ${totalBudget}* [64]. Esse valor mostra o tamanho do investimento atual em projetos.\n- **ROI Projetado (Total):** soma do retorno esperado de todos projetos, usando fator estimado. Ex.: *R$ ${totalROI}* projetados, considerando 3,47x de ROI em cada projeto [14].\n- **ROI Médio (%):** média de retorno esperado. No caso do fator uniforme, é *347%* [46] (indicando que, coletivamente, espera-se recuperar 3,47 vezes o investido). Esses números servem para avaliar se o portfólio está alinhado com metas de retorno e para dar visibilidade financeira à diretoria."
    },
    {
      "id": "distribution_status",
      "type": "text",
      "x": 1540,
      "y": 1640,
      "width": 360,
      "height": 130,
      "color": "1",
      "text": "🏷 **Distribuição por Status**\nOs projetos atuais se dividem entre diferentes status (fases). A query agrupa e conta projetos por status [22]. Por exemplo:\n- **Ideia:** 2 projetos\n- **Em Planejamento:** 3 projetos\n- **Em Execução:** 5 projetos\n- **Concluído:** 0 (pois concluídos não estão no recorte atual)\nEssa distribuição revela gargalos ou acúmulos. Se muitos projetos estão parados em \"Planejamento\", pode indicar demora em iniciar execuções. Se poucos estão em fase final, pode preocupar quanto a entregas. Acompanhar a mudança dessa distribuição ao longo do tempo sinaliza progresso do pipeline."
    },
    {
      "id": "distribution_owner",
      "type": "text",
      "x": 1540,
      "y": 1770,
      "width": 360,
      "height": 130,
      "color": "1",
      "text": "  **Distribuição por Responsável**\nAgrupa os projetos por *owner* (gerente responsável) e conta quantos projetos cada pessoa lidera [25]. Exemplo de output:\n- Pedro Vitor: 5 projetos\n- Ana Souza: 2 projetos\n- Cláudia M.: 1 projeto\nNesse cenário, Pedro Vitor concentra muitos projetos. Esse insight permitiu identificar potencial sobrecarga gerencial. De fato, no vault real, um insight destacado foi \"Pedro Vitor gerencia X projetos\" [65]. Caso a disparidade seja grande, ações podem ser tomadas (redistribuir projetos). Também serve para planejar sucessão e mentoring – se um gestor único concentra muito conhecimento de projetos."
    },
    {
      "id": "distribution_department",
      "type": "text",
      "x": 1540,
```

```
      "y": 1900,
      "width": 360,
      "height": 150,
      "color": "1",
      "text": "  **Capacidade por Departamento**\nA consolidação por
departamento mostra como as pessoas (e suas horas) se distribuem nas áreas da
empresa [35]. Exemplo:\n- TI: 5 pessoas, 200h/sem capacidade, 180h alocadas
(90% util.)\n- Marketing: 3 pessoas, 120h cap., 60h alocadas (50% util.)\n-
Vendas: 2 pessoas, 80h cap., 40h aloc. (50% util.)\nNesse exemplo, TI está
com alta utilização (90%), enquanto Marketing e Vendas têm folga. Isso
orienta decisões de alocação: talvez TI precise de contratações ou de
transferir parte do trabalho para outras áreas se possível. Também ajuda a
priorizar projetos inter-departamentais conforme disponibilidade."
    },
    {
      "id": "alert_overallocated",
      "type": "text",
      "x": 1540,
      "y": 2050,
      "width": 360,
      "height": 120,
      "color": "1",
      "text": "  **Pessoas Sobrecarregadas (>80%)**\nO sistema identifica
indivíduos com carga acima do limite saudável [37]. Por exemplo, se João está
com 45h/sem alocadas de 40h (112%), ou Maria com 35h de 40h (87.5%). Esses
nomes aparecem na lista de *Alertas de Capacidade - Sobrecarregados*. São
sinais vermelhos para gestão: é preciso redistribuir tarefas ou negociar
prazos. No vault, após ver esses alertas, houve *Recomendação*:
\"Redistribuir carga de Pedro Vitor para outros membros\" [41], ilustrando como
o insight vira ação."
    },
    {
      "id": "alert_underallocated",
      "type": "text",
      "x": 1540,
      "y": 2180,
      "width": 360,
      "height": 120,
      "color": "1",
      "text": "  **Pessoas Subutilizadas (<30%)**\nSimilarmente, a query
aponta quem está com pouca alocação [38]. Exemplo: se um analista tem só 10h de
40h alocadas (25%), aparece na lista. Isso pode indicar capacidade não
aproveitada – oportunidade de incluir essa pessoa em novos projetos ou
atividades de melhoria. A Recomendação associada foi: \"Aproveitar horas
livres para novos projetos\" [66]. Esse insight ajuda a empresa a usar melhor
seus recursos humanos, evitando que talentos fiquem ociosos."
    },
    {
      "id": "high_value_projects_metric",
      "type": "text",
      "x": 1540,
```

```
      "y": 1480,
      "width": 360,
      "height": 130,
      "color": "1",
      "text": "  **Projetos de Alto Valor**\nSão destacados projetos com budget elevado, e.g. >= R$ 50k [67] . Suponha que 3 projetos passam esse critério. Isso alerta a direção sobre quais iniciativas envolvem mais recursos financeiros – naturalmente requerendo acompanhamento mais próximo. Muitas vezes, projetos de alto valor também têm maior impacto ou risco, então essa lista serve para priorização gerencial e possíveis análises de ROI individual. No vault, isso foi ligado à recomendação: \"Foco em ROI – Projetos >50k têm maior potencial de retorno\" [66] ."
    },
    {
      "id": "insight_projects_concentration",
      "type": "text",
      "x": 1320,
      "y": 1580,
      "width": 360,
      "height": 110,
      "color": "1",
      "text": "  **Insight: Concentração de Projetos**\nUm insight personalizado gerado: *\"Pedro Vitor gerencia X projetos\"* [65] . Esse achado evidenciou a centralização de projetos críticos em um único gestor. Embora fosse codificado fixamente para Pedro, ele exemplifica como o sistema pode trazer à tona informações relevantes para equilíbrio de carga (neste caso, não carga de horas, mas de responsabilidade/gestão). Ações tomadas incluíram redistribuir projetos e planejar sucessores para reduzir risco de depender demais de uma pessoa."
    },
    {
      "id": "insight_free_capacity",
      "type": "text",
      "x": 1320,
      "y": 1700,
      "width": 360,
      "height": 110,
      "color": "1",
      "text": "  **Insight: Capacidade Disponível**\nOutro insight exibido: *\"${horasLivres}h livres por semana\"* na equipe [45] . Ou seja, a soma das horas não alocadas de todos. Isso complementa a % média – ex: 80h livres equivalem a 2 FTEs disponíveis. Tornar explícito em horas ajuda no planejamento: a liderança sabe quanta carga extra poderia ser absorvida. Se esse número cai a quase 0h, significa saturação (e possivelmente justifica contratar ou re-priorizar). Se é muito alto, talvez a empresa possa assumir mais projetos ou está subutilizando pessoal (reforçando o insight de subutilização)."
    },
    {
      "id": "insight_avg_roi",
      "type": "text",
```

```
      "x": 1320,
      "y": 1820,
      "width": 360,
      "height": 110,
      "color": "1",
      "text":
"  **Insight: ROI Médio Projetado**\nNo dashboard foi mostrado: *\"347% com
receita projetada de R$ ${totalROI}\"* 46. Ou seja, em média espera-se
triplicar investimentos (347% retorno). Esse insight resume a eficiência
financeira esperada dos projetos. Se a empresa tem metas de ROI, pode
comparar esse 347% com a meta (ex: meta 400% – então 347% estaria abaixo).
Também ajuda a comunicar para stakeholders: \"Nosso portfólio atual tem ROI
médio estimado de ~3,5x\". Importante: esse número deriva do fator assumido,
então serve mais como referência macro do que como previsão exata."
    }
  ],
  "edges": [
    {
      "id": "e1",
      "fromNode": "fundamentals_main",
      "fromSide": "right",
      "toNode": "integration_main",
      "toSide": "left",
      "color": "4",
      "label": "Base conceitual integra no todo"
    },
    {
      "id": "e2",
      "fromNode": "fundamentals_main",
      "fromSide": "bottom",
      "toNode": "obsidian_vault",
      "toSide": "top",
      "color": "4"
    },
    {
      "id": "e3",
      "fromNode": "fundamentals_main",
      "fromSide": "bottom",
      "toNode": "yaml_metadata",
      "toSide": "top",
      "color": "4"
    },
    {
      "id": "e4",
      "fromNode": "fundamentals_main",
      "fromSide": "bottom",
      "toNode": "dataview_concept",
      "toSide": "top",
      "color": "4"
    },
    {
```

```
    "id": "e5",
    "fromNode": "obsidian_vault",
    "fromSide": "right",
    "toNode": "folders_vs_fields",
    "toSide": "left",
    "color": "4",
    "label": "Organização"
  },
  {
    "id": "e6",
    "fromNode": "yaml_metadata",
    "fromSide": "right",
    "toNode": "folders_vs_fields",
    "toSide": "left",
    "color": "4",
    "label": "Metadados vs Pastas"
  },
  {
    "id": "e7",
    "fromNode": "yaml_metadata",
    "fromSide": "right",
    "toNode": "data_consistency",
    "toSide": "left",
    "color": "4",
    "label": "Campos YAML padronizados"
  },
  {
    "id": "e8",
    "fromNode": "dataview_concept",
    "fromSide": "right",
    "toNode": "dql_basics",
    "toSide": "left",
    "color": "4",
    "label": "Sintaxe"
  },
  {
    "id": "e9",
    "fromNode": "dataview_concept",
    "fromSide": "right",
    "toNode": "dv_limitations",
    "toSide": "left",
    "color": "4",
    "label": "Limitações"
  },
  {
    "id": "e10",
    "fromNode": "dataview_concept",
    "fromSide": "bottom",
    "toNode": "dataviewjs_note",
    "toSide": "top",
    "color": "4",
```

```
    "label": "JS avançado"
  },
  {
    "id": "e11",
    "fromNode": "folders_vs_fields",
    "fromSide": "right",
    "toNode": "folder_structure",
    "toSide": "left",
    "color": "6",
    "label": "No vault"
  },
  {
    "id": "e12",
    "fromNode": "data_consistency",
    "fromSide": "right",
    "toNode": "data_quality_governance",
    "toSide": "left",
    "color": "6",
    "label": "Validação"
  },
  {
    "id": "e13",
    "fromNode": "data_consistency",
    "fromSide": "right",
    "toNode": "standard_templates",
    "toSide": "left",
    "color": "6",
    "label": "Assegurar padrão"
  },
  {
    "id": "e14",
    "fromNode": "dataviewjs_note",
    "fromSide": "right",
    "toNode": "automation_flows",
    "toSide": "left",
    "color": "2",
    "label": "Scripts e agentes"
  },
  {
    "id": "e15",
    "fromNode": "dv_limitations",
    "fromSide": "right",
    "toNode": "automation_flows",
    "toSide": "left",
    "color": "2",
    "label": "Complementar com outras ferramentas"
  },
  {
    "id": "e16",
    "fromNode": "applications_main",
    "fromSide": "left",
```

```
      "toNode": "integration_main",
      "toSide": "right",
      "color": "2",
      "label": "Aplicações integradas"
    },
    {
      "id": "e17",
      "fromNode": "applications_main",
      "fromSide": "bottom",
      "toNode": "dashboard_exec",
      "toSide": "top",
      "color": "2"
    },
    {
      "id": "e18",
      "fromNode": "applications_main",
      "fromSide": "bottom",
      "toNode": "project_pipeline",
      "toSide": "top",
      "color": "2"
    },
    {
      "id": "e19",
      "fromNode": "applications_main",
      "fromSide": "bottom",
      "toNode": "team_capacity_dashboard",
      "toSide": "top",
      "color": "2"
    },
    {
      "id": "e20",
      "fromNode": "applications_main",
      "fromSide": "bottom",
      "toNode": "financial_analysis",
      "toSide": "top",
      "color": "2"
    },
    {
      "id": "e21",
      "fromNode": "dashboard_exec",
      "fromSide": "right",
      "toNode": "projects_overview",
      "toSide": "left",
      "color": "1",
      "label": "Métrica"
    },
    {
      "id": "e22",
      "fromNode": "dashboard_exec",
      "fromSide": "right",
      "toNode": "team_overview",
```

```
      "toSide": "left",
      "color": "1",
      "label": "Indicador"
    },
    {
      "id": "e23",
      "fromNode": "dashboard_exec",
      "fromSide": "right",
      "toNode": "financial_totals",
      "toSide": "left",
      "color": "1",
      "label": "Indicador"
    },
    {
      "id": "e24",
      "fromNode": "project_pipeline",
      "fromSide": "right",
      "toNode": "distribution_status",
      "toSide": "left",
      "color": "1",
      "label": "Contagem por status"
    },
    {
      "id": "e25",
      "fromNode": "project_pipeline",
      "fromSide": "right",
      "toNode": "distribution_owner",
      "toSide": "left",
      "color": "1",
      "label": "Contagem por owner"
    },
    {
      "id": "e26",
      "fromNode": "team_capacity_dashboard",
      "fromSide": "right",
      "toNode": "capacity_allocation_totals",
      "toSide": "left",
      "color": "1",
      "label": "Horas"
    },
    {
      "id": "e27",
      "fromNode": "team_capacity_dashboard",
      "fromSide": "right",
      "toNode": "distribution_department",
      "toSide": "left",
      "color": "1",
      "label": "Agrupado depto"
    },
    {
      "id": "e28",
```

```
      "fromNode": "team_capacity_dashboard",
      "fromSide": "right",
      "toNode": "alert_overallocated",
      "toSide": "left",
      "color": "1",
      "label": "Alerta"
    },
    {
      "id": "e29",
      "fromNode": "team_capacity_dashboard",
      "fromSide": "right",
      "toNode": "alert_underallocated",
      "toSide": "left",
      "color": "1",
      "label": "Alerta"
    },
    {
      "id": "e30",
      "fromNode": "financial_analysis",
      "fromSide": "right",
      "toNode": "high_value_projects_metric",
      "toSide": "left",
      "color": "1",
      "label": "Alto Budget"
    },
    {
      "id": "e31",
      "fromNode": "financial_analysis",
      "fromSide": "right",
      "toNode": "financial_totals",
      "toSide": "left",
      "color": "1",
      "label": "Somas"
    },
    {
      "id": "e32",
      "fromNode": "alerts_system",
      "fromSide": "right",
      "toNode": "alert_overallocated",
      "toSide": "left",
      "color": "1",
      "label": "Sobrecarregados"
    },
    {
      "id": "e33",
      "fromNode": "alerts_system",
      "fromSide": "right",
      "toNode": "alert_underallocated",
      "toSide": "left",
      "color": "1",
      "label": "Subutilizados"
```

```
    },
    {
      "id": "e34",
      "fromNode": "alerts_system",
      "fromSide": "right",
      "toNode": "high_value_projects_metric",
      "toSide": "left",
      "color": "1",
      "label": "Projetos $ alto"
    },
    {
      "id": "e35",
      "fromNode": "task_management",
      "fromSide": "left",
      "toNode": "popup_navigation",
      "toSide": "right",
      "color": "2",
      "label": "Integra UX"
    },
    {
      "id": "e36",
      "fromNode": "task_management",
      "fromSide": "left",
      "toNode": "automation_flows",
      "toSide": "right",
      "color": "2",
      "label": "Automação tasks"
    },
    {
      "id": "e37",
      "fromNode": "popup_navigation",
      "fromSide": "left",
      "toNode": "dashboard_exec",
      "toSide": "right",
      "color": "2",
      "label": "Previews in vault"
    },
    {
      "id": "e38",
      "fromNode": "collab_sharing",
      "fromSide": "left",
      "toNode": "integration_netlify",
      "toSide": "right",
      "color": "6",
      "label": "Publicação"
    },
    {
      "id": "e39",
      "fromNode": "automation_flows",
      "fromSide": "left",
      "toNode": "agents_automation",
```

```
      "toSide": "right",
      "color": "5",
      "label": "Bots"
    },
    {
      "id": "e40",
      "fromNode": "methodology_main",
      "fromSide": "right",
      "toNode": "integration_main",
      "toSide": "bottom",
      "color": "6",
      "label": "Processos permitem integração"
    },
    {
      "id": "e41",
      "fromNode": "methodology_main",
      "fromSide": "bottom",
      "toNode": "standard_templates",
      "toSide": "top",
      "color": "6"
    },
    {
      "id": "e42",
      "fromNode": "methodology_main",
      "fromSide": "bottom",
      "toNode": "folder_structure",
      "toSide": "top",
      "color": "6"
    },
    {
      "id": "e43",
      "fromNode": "methodology_main",
      "fromSide": "bottom",
      "toNode": "version_control",
      "toSide": "top",
      "color": "6"
    },
    {
      "id": "e44",
      "fromNode": "methodology_main",
      "fromSide": "bottom",
      "toNode": "data_quality_governance",
      "toSide": "top",
      "color": "6"
    },
    {
      "id": "e45",
      "fromNode": "standard_templates",
      "fromSide": "right",
      "toNode": "yaml_metadata",
      "toSide": "bottom",
```

```
      "color": "6",
      "label": "Preenche frontmatter"
    },
    {
      "id": "e46",
      "fromNode": "folder_structure",
      "fromSide": "right",
      "toNode": "folders_vs_fields",
      "toSide": "bottom",
      "color": "6",
      "label": "Mapeamento"
    },
    {
      "id": "e47",
      "fromNode": "version_control",
      "fromSide": "right",
      "toNode": "integration_netlify",
      "toSide": "left",
      "color": "5",
      "label": "Base para Publish"
    },
    {
      "id": "e48",
      "fromNode": "data_quality_governance",
      "fromSide": "right",
      "toNode": "data_consistency",
      "toSide": "bottom",
      "color": "6",
      "label": "Assegurar integridade"
    },
    {
      "id": "e49",
      "fromNode": "training_onboarding",
      "fromSide": "top",
      "toNode": "methodology_main",
      "toSide": "bottom",
      "color": "6"
    },
    {
      "id": "e50",
      "fromNode": "training_onboarding",
      "fromSide": "right",
      "toNode": "collab_sharing",
      "toSide": "left",
      "color": "6",
      "label": "Aderência do time"
    },
    {
      "id": "e51",
      "fromNode": "maintenance_evolution",
      "fromSide": "top",
```

```
      "toNode": "methodology_main",
      "toSide": "bottom",
      "color": "6"
    },
    {
      "id": "e52",
      "fromNode": "maintenance_evolution",
      "fromSide": "right",
      "toNode": "integration_scalability",
      "toSide": "left",
      "color": "5",
      "label": "Ajustes"
    },
    {
      "id": "e53",
      "fromNode": "results_main",
      "fromSide": "left",
      "toNode": "integration_main",
      "toSide": "bottom",
      "color": "1",
      "label": "Insights para integração"
    },
    {
      "id": "e54",
      "fromNode": "results_main",
      "fromSide": "bottom",
      "toNode": "projects_overview",
      "toSide": "top",
      "color": "1"
    },
    {
      "id": "e55",
      "fromNode": "results_main",
      "fromSide": "bottom",
      "toNode": "distribution_status",
      "toSide": "top",
      "color": "1"
    },
    {
      "id": "e56",
      "fromNode": "results_main",
      "fromSide": "bottom",
      "toNode": "capacity_allocation_totals",
      "toSide": "top",
      "color": "1"
    },
    {
      "id": "e57",
      "fromNode": "results_main",
      "fromSide": "bottom",
      "toNode": "alert_overallocated",
```

```
      "toSide": "top",
      "color": "1"
    },
    {
      "id": "e58",
      "fromNode": "projects_overview",
      "fromSide": "right",
      "toNode": "insight_projects_concentration",
      "toSide": "left",
      "color": "1",
      "label": "Contexto"
    },
    {
      "id": "e59",
      "fromNode": "team_overview",
      "fromSide": "right",
      "toNode": "insight_free_capacity",
      "toSide": "left",
      "color": "1",
      "label": "Mostra"
    },
    {
      "id": "e60",
      "fromNode": "financial_totals",
      "fromSide": "right",
      "toNode": "insight_avg_roi",
      "toSide": "left",
      "color": "1",
      "label": "Resumo"
    },
    {
      "id": "e61",
      "fromNode": "distribution_owner",
      "fromSide": "right",
      "toNode": "insight_projects_concentration",
      "toSide": "left",
      "color": "1",
      "label": "Pedro Vitor"
    },
    {
      "id": "e62",
      "fromNode": "capacity_allocation_totals",
      "fromSide": "right",
      "toNode": "insight_free_capacity",
      "toSide": "left",
      "color": "1",
      "label": "80h livres"
    },
    {
      "id": "e63",
      "fromNode": "alert_overallocated",
```

```
      "fromSide": "right",
      "toNode": "insight_projects_concentration",
      "toSide": "bottom",
      "color": "1",
      "label": "Impacto nos gestores"
    },
    {
      "id": "e64",
      "fromNode": "alert_overallocated",
      "fromSide": "right",
      "toNode": "insight_free_capacity",
      "toSide": "bottom",
      "color": "1",
      "label": "Pouca folga"
    },
    {
      "id": "e65",
      "fromNode": "alert_underallocated",
      "fromSide": "right",
      "toNode": "insight_free_capacity",
      "toSide": "bottom",
      "color": "1",
      "label": "Muita folga"
    },
    {
      "id": "e66",
      "fromNode": "high_value_projects_metric",
      "fromSide": "right",
      "toNode": "insight_avg_roi",
      "toSide": "bottom",
      "color": "1",
      "label": "Concentra ROI"
    }
  ]
}
```

---

1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 49 50 51 52 53 55 56 57 60 61 62 63 64 65 66 67 compilado vault.md
file://file-ABhnrgB7rsZ2ZurL3Msq5L

2 How to have several sources under "from" in dataview - Help - Obsidian Forum
https://forum.obsidian.md/t/how-to-have-several-sources-under-from-in-dataview/51531

18 Query Types - Dataview - GitHub Pages
https://blacksmithgu.github.io/obsidian-dataview/queries/query-types/

47 48 Functions - Dataview
https://blacksmithgu.github.io/obsidian-dataview/reference/functions/

54 Persist the cache to improve startup performance #1064 - GitHub
https://github.com/blacksmithgu/obsidian-dataview/issues/1064

58 Dataview plugin - Fork My Brain
https://notes.nicolevanderhoeven.com/obsidian-playbook/Obsidian+Plugins/Community+Plugins/dataview/Dataview+plugin

59 Obsidian Dataview: Build your Vault as a Database | by Esteban Thilliez | The Obsidianist | Medium
https://medium.com/the-obsidianist/obsidian-dataview-build-your-vault-as-a-database-fb3920f8cd79