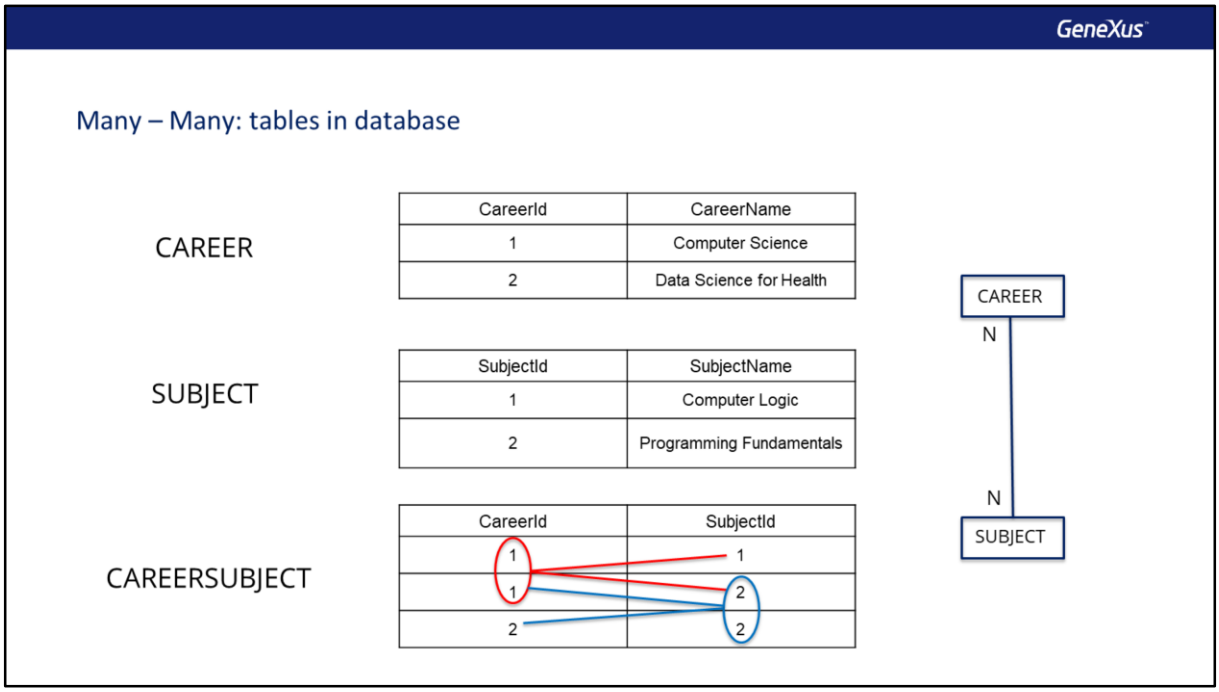
Desenho de transações e Normalização

Uma visão integradora

GeneXus™

Neste vídeo vamos tentar analisar diferentes temas relacionados com o desenho de transações e como as decisões que tomamos se refletem nas estruturas da base de dados criadas, ou na funcionalidade da aplicação.

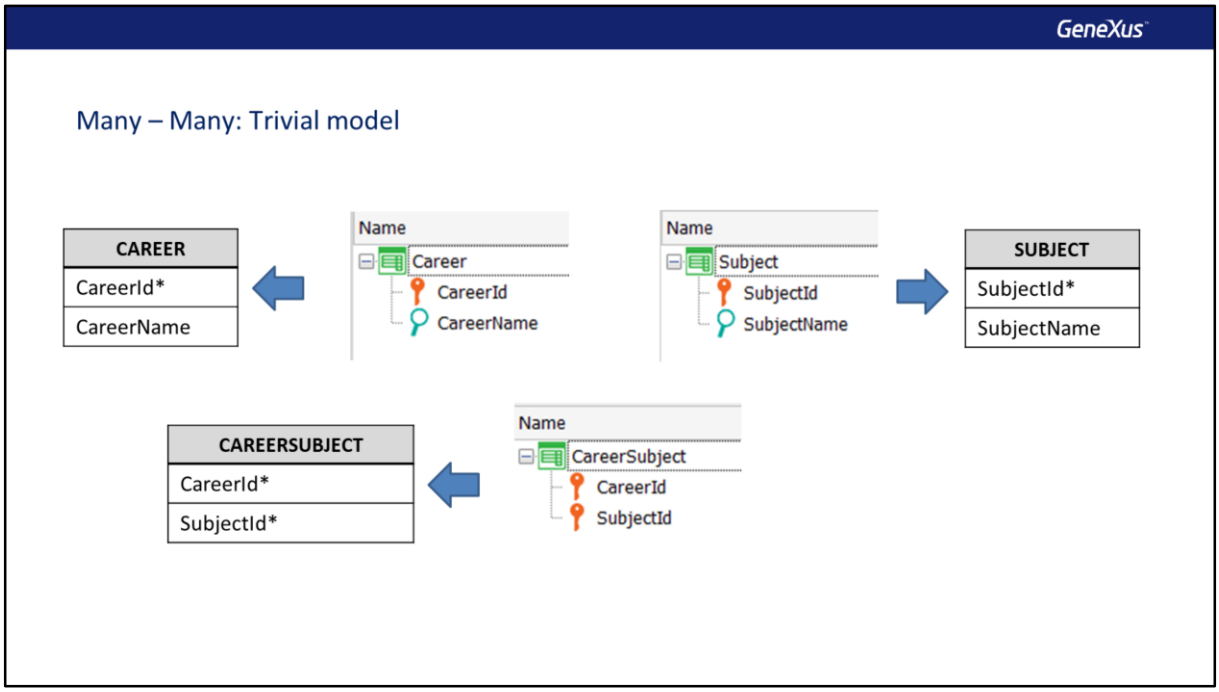
Por esta razão, ao contrário dos vídeos anteriores sobre o tema onde se seguiu o guia do desenvolvimento de um exemplo, neste caso iremos abordar casos específicos que nos permitem analisar diferentes situações práticas que podem ser úteis na construção da nossa solução.



Para representar uma relação muitos para muitos entre duas entidades em uma base de dados relacional, são utilizadas três tabelas; uma para cada entidade e uma terceira (também chamada de tabela de relação) que contém os identificadores das tabelas anteriores formando uma chave composta.

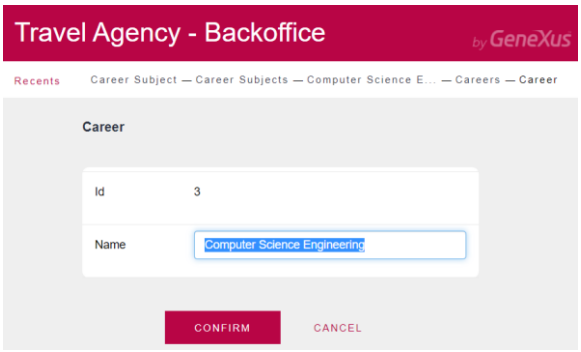
Vamos pegar como exemplo o caso da realidade de uma universidade, onde cada carreira tem muitas matérias e cada matéria pode estar em muitas carreiras. Para representar esta relação, teremos uma tabela Career, uma tabela Subject e uma tabela CareerSubject que tem como chave as chaves das tabelas anteriores formando uma chave composta.

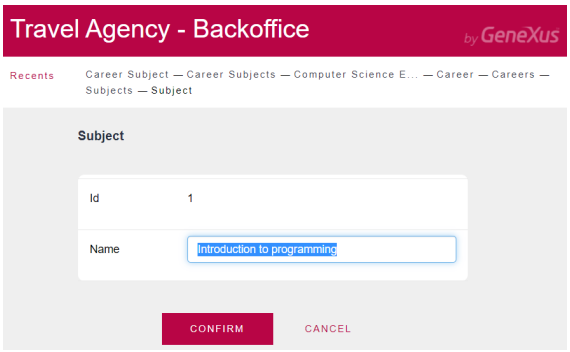
Se analisarmos os dados, vemos que a carreira 1 contém as matérias 1 e 2, mas que por sua vez a matéria 2 está na carreira 1 e na carreira 2, então este modelo efetivamente nos permite representar uma relação de muitos para muitos, entre carreiras e matérias.

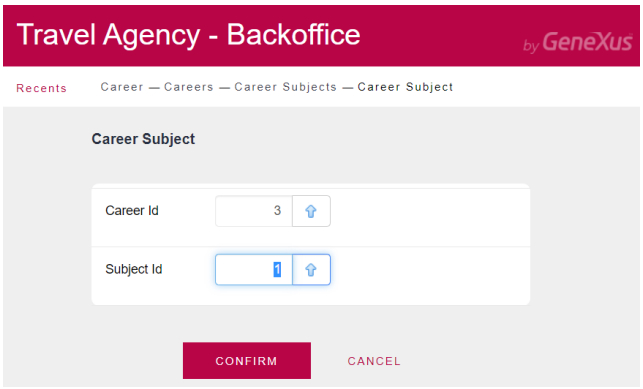


Em GeneXus não usamos tabelas para modelar a realidade, mas sim transações. Uma solução trivial para modelar a relação entre carreiras e matérias de forma que GeneXus gere as três tabelas que necessitamos, é criar transações com a mesma estrutura das tabelas.

Como as três transações são planas, ou seja, não há subníveis, serão criadas as tabelas conforme o esperado, de forma que podemos garantir que este modelo efetivamente representa uma relação muitos para muitos entre carreiras e matérias.

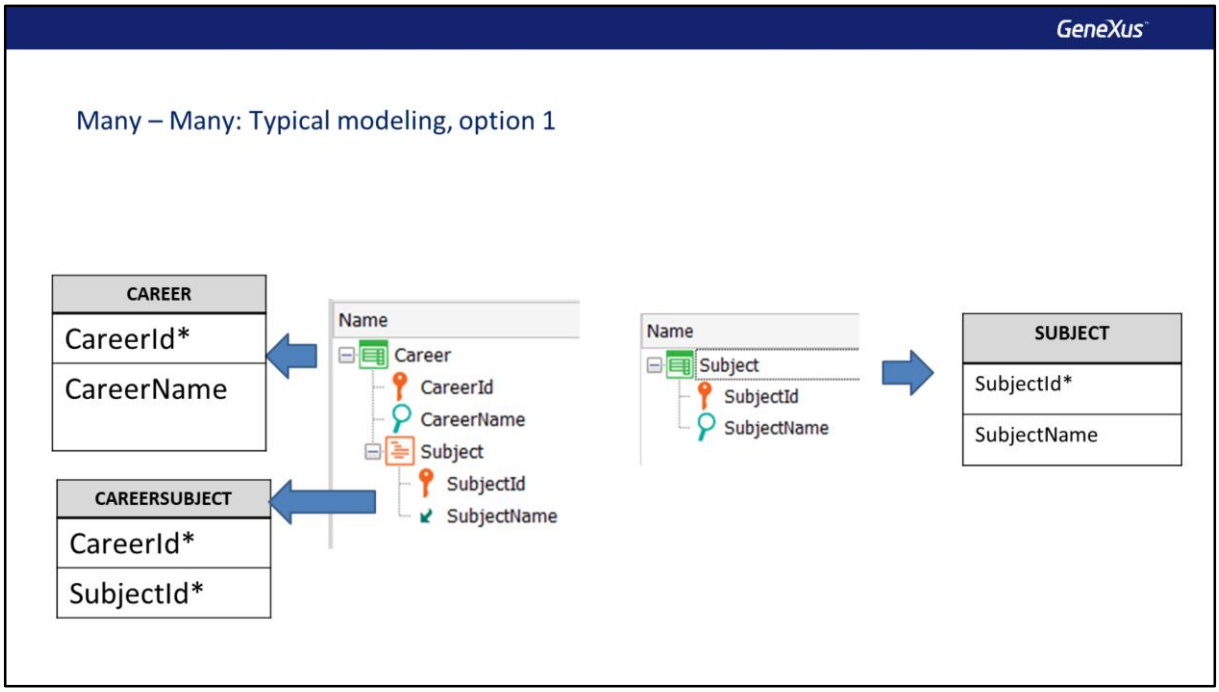






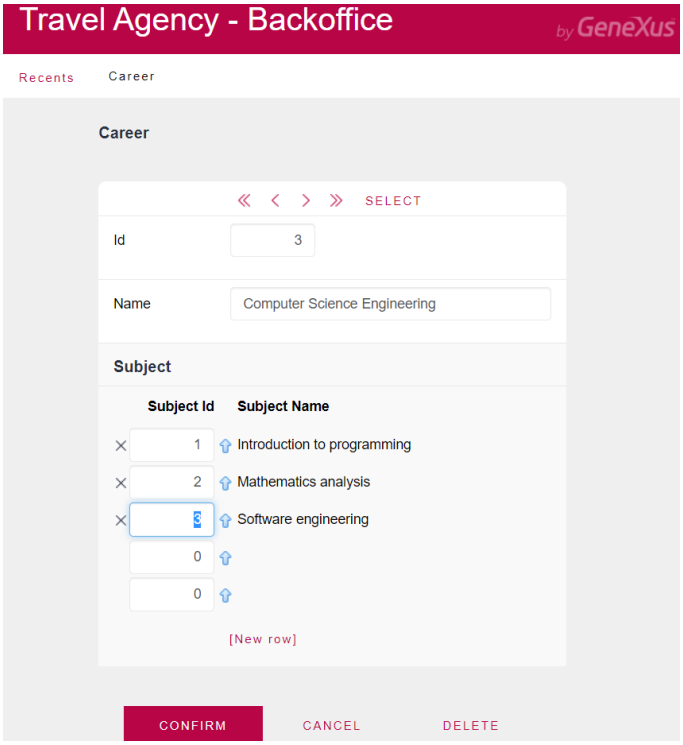
Porém, se considerarmos a interface de usuário na hora de inserir os dados, teríamos 3 telas. As carreiras e as matérias iríamos inserir sem problemas, mas depois é muito pouco amigável ter que inserir os pares de identificadores de carreira e matéria para estabelecer quais carreiras correspondem a qual matéria e vice-versa.

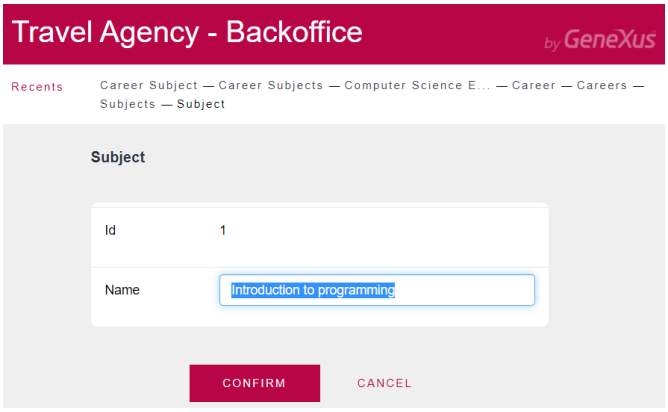
Portanto, embora este desenho cumpra com a modelagem da relação muitos para muitos, não seria o recomendado. É preferível que em uma tela possamos inserir os dados de uma carreira e depois em um grid todas as matérias dessa carreira, ou vice-versa....



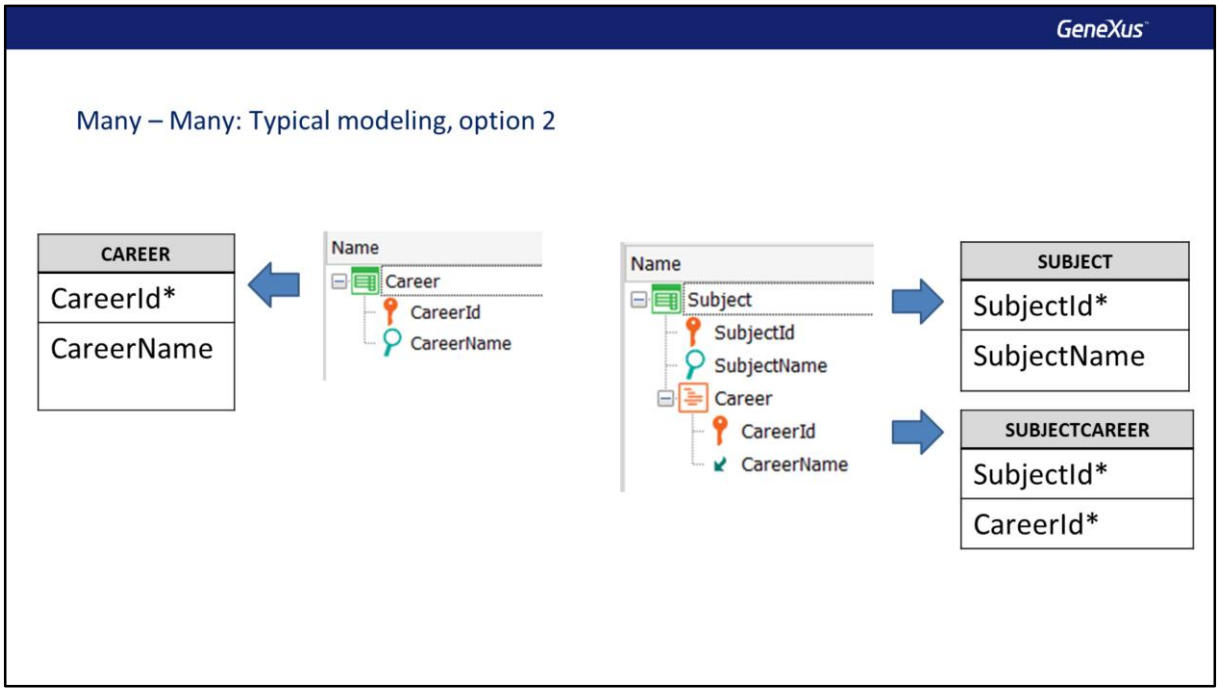
Para obter uma tela onde para uma carreira possam ser inseridas todas as suas matérias, podemos criar uma transação Career de dois níveis, onde o nível aninhado são as matérias. Com essa transação apenas, modelaríamos uma relação 1 para muitos entre carreiras e matérias, para que seja uma relação muitos para muitos, adicionamos uma segunda transação Subject.

Se olharmos as tabelas que GeneXus cria, veremos que efetivamente modelamos uma relação muitos para muitos entre carreiras e matérias, já que obtemos as mesmas três tabelas que vimos anteriormente.





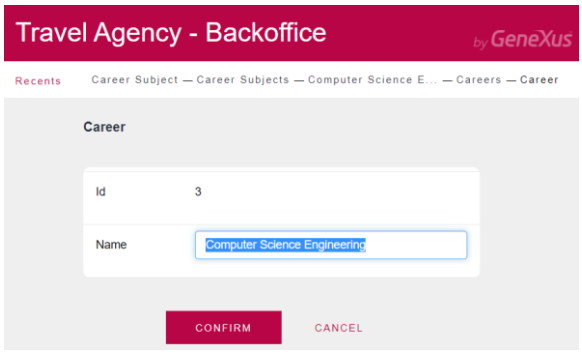
Em execução vemos que nosso modelo priorizou a abordagem de inserir as matérias, para depois carregar as matérias de cada carreira.

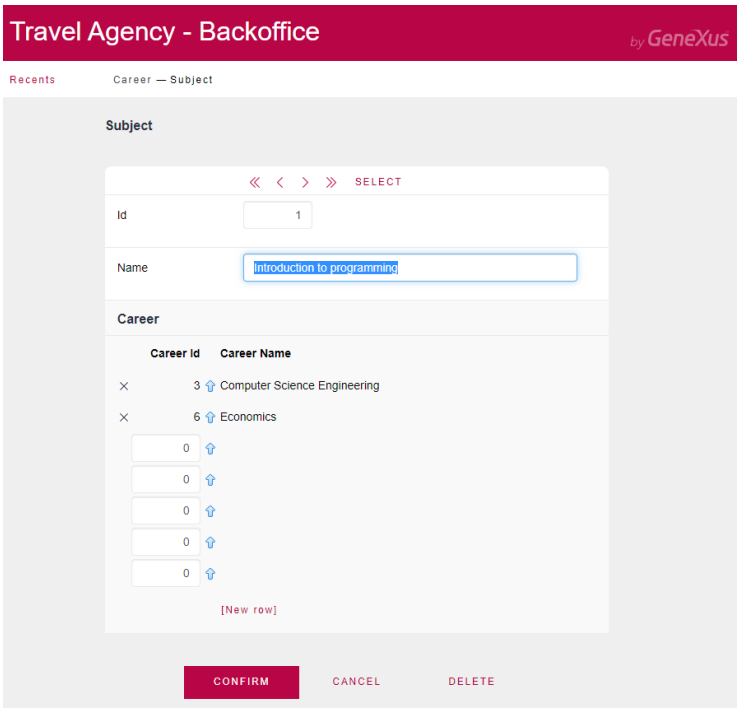


Agora, se para o mesmo caso da relação muitos para muitos entre matérias e carreiras, nos fosse solicitado uma tela que para cada matéria inseríssemos as carreiras a que pertence, então criaríamos uma transação Subject de dois níveis, onde o segundo nível corresponde às carreiras.

Obviamente também criamos uma transação Career para que seja mantida a relação muitos para muitos entre as entidades.

Se olharmos as tabelas que serão criadas, verificamos que são exatamente as mesmas dos exemplos anteriores, então temos certeza de que modelamos uma relação muitos para muitos entre matérias e carreiras.

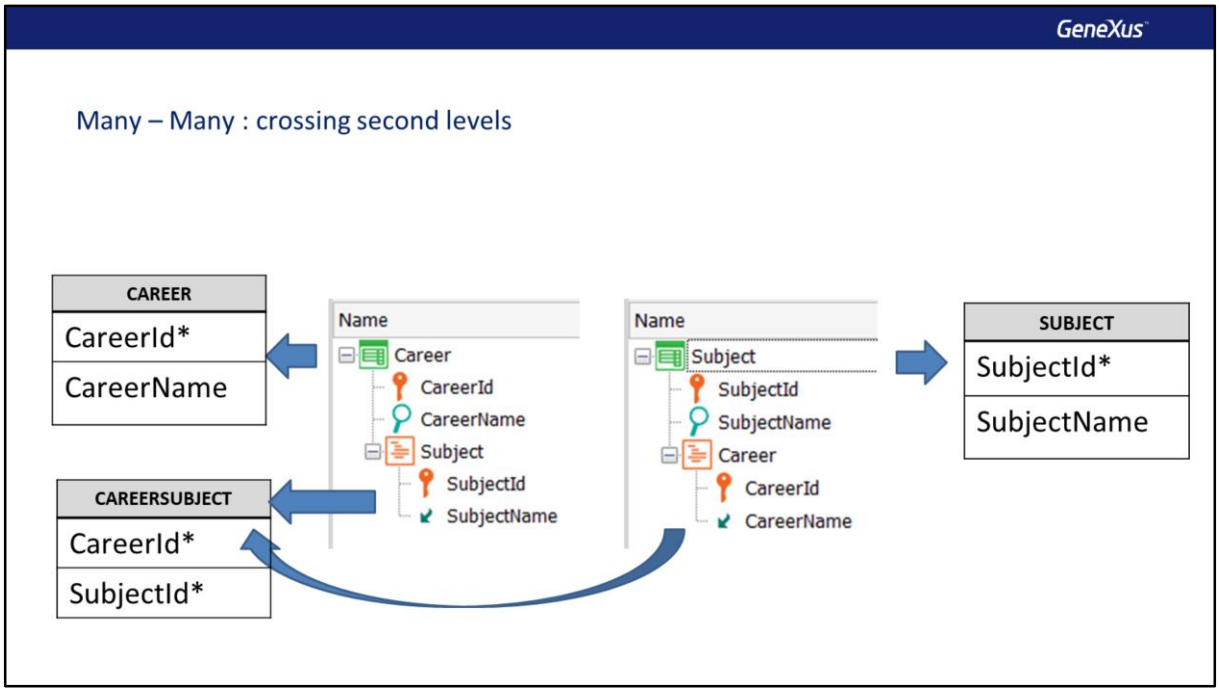




Em execución verificamos que agora o foco é que primeiro inserimos as carreiras e depois para cada matéria, inserimos as carreiras que pertencem a ela.

Então para modelar uma relação muitos para muitos, basta com duas transações, uma com dois níveis, e para a entidade que colocamos no segundo nível, criamos também uma transação separada.

Do ponto de vista da relação muitos para muitos, não importa qual entidade colocamos no segundo nível, só tem relevância na hora de inserir os dados.



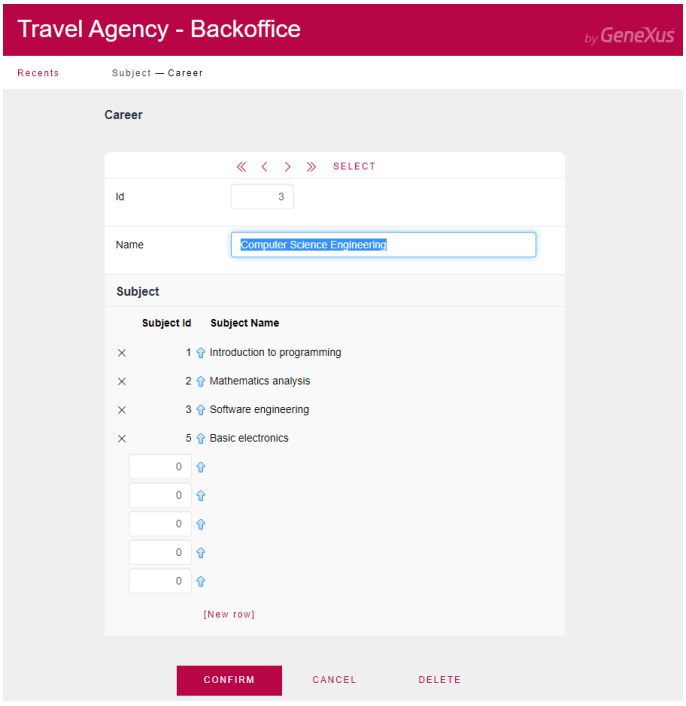
Agora, o que aconteceria se nos pedissem que em uma tela seja possível inserir uma carreira e para essa carreira todas as matérias que contém e ao mesmo tempo outra tela onde seja possível inserir uma matéria e todas as carreiras que pertencem a ela?
Seguindo o raciocínio anterior, deveríamos criar duas transações de dois níveis, uma chamada Career com o segundo nível Subject e outra chamada Subject com o segundo nível Career.

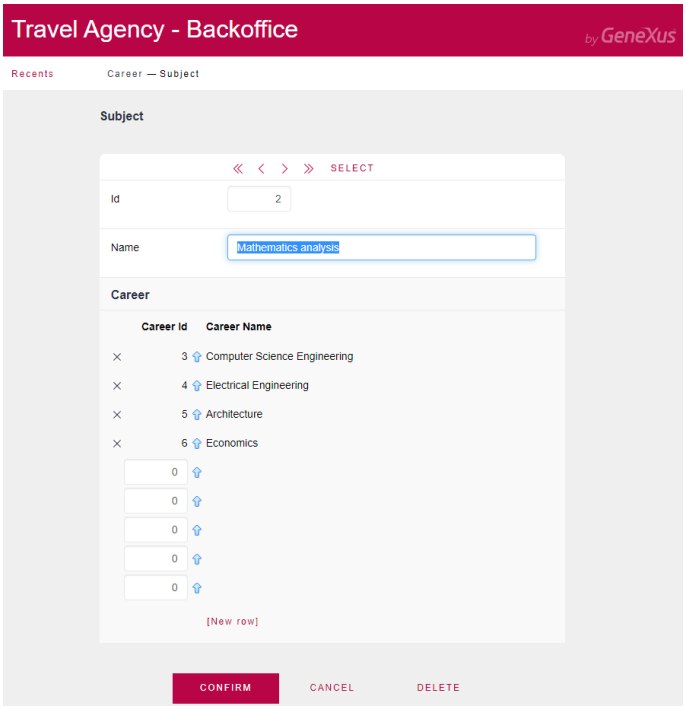
Mas que relação estaríamos modelando neste caso?
Para saber, escrevemos as tabelas que GeneXus criará. Sabemos que a partir da transação Career será criada uma tabela CAREER com a estrutura do primeiro nível da transação e que a partir do segundo nível será criada uma tabela CAREERSUBJECT que, por ser o atributo SubjectName inferido, a tabela conterá apenas os atributos identificadores do primeiro e do segundo nível, formando uma chave composta.

Se analisarmos as tabelas que serão criadas a partir da transação de dois níveis Subject, verificamos que a partir do primeiro nível será criada uma tabela SUBJECT com a mesma estrutura que o cabeçalho da transação Subject e a partir do segundo nível uma tabela que assumimos que seria chamada de SUBJECTCAREER contendo apenas uma chave composta por SubjectId e CarrerId, já que o atributo CareerName é inferido.

Mas GeneXus já criou uma tabela exatamente com essa estrutura, a tabela CAREERSUBJECT, portanto não cria outra e o resultado final deste modelo são as mesmas três tabelas que obtivemos antes e que sabemos que correspondem a uma relação muitos para muitos entre carreiras e matérias.

Portanto, se criarmos duas transações de dois níveis e cruzarmos as entidades, ou seja, em uma colocarmos como segundo nível o que na outra é o primeiro nível e vice-versa, estaremos modelando uma relação muitos para muitos.



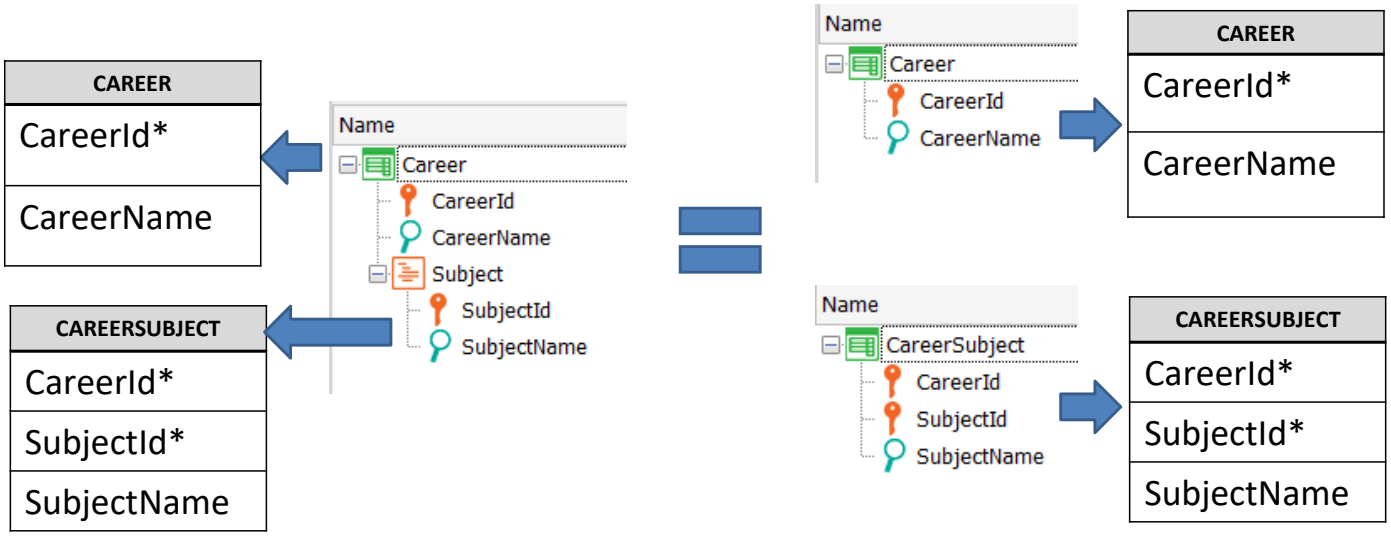


No entanto, com este modelo, no momento da inserção de dados, temos algumas limitações. Por exemplo, se queremos inserir uma carreira, podemos inserir seu cabeçalho, mas, devido ao controle automático de integridade referencial, não podemos inserir as matérias que correspondem a ela, porque ainda não foi inserida nenhuma matéria.

O mesmo acontece se abrirmos primeiro a transação Subject, só poderemos inserir os dados de cada matéria, mas não as carreiras que pertencem a ela.

Então devemos primeiro inserir em uma delas todos os cabeçalhos sem linhas, e depois ir para a outra transação e inserir o cabeçalho e depois podemos inserir as linhas correspondentes.

Flattening the design



Uma transação de vários níveis pode ser decomposta em transações de nível único. A esta operação dizemos nivelar o modelo, pois todas as transações passarão a ser planas, ou seja, sem subníveis.

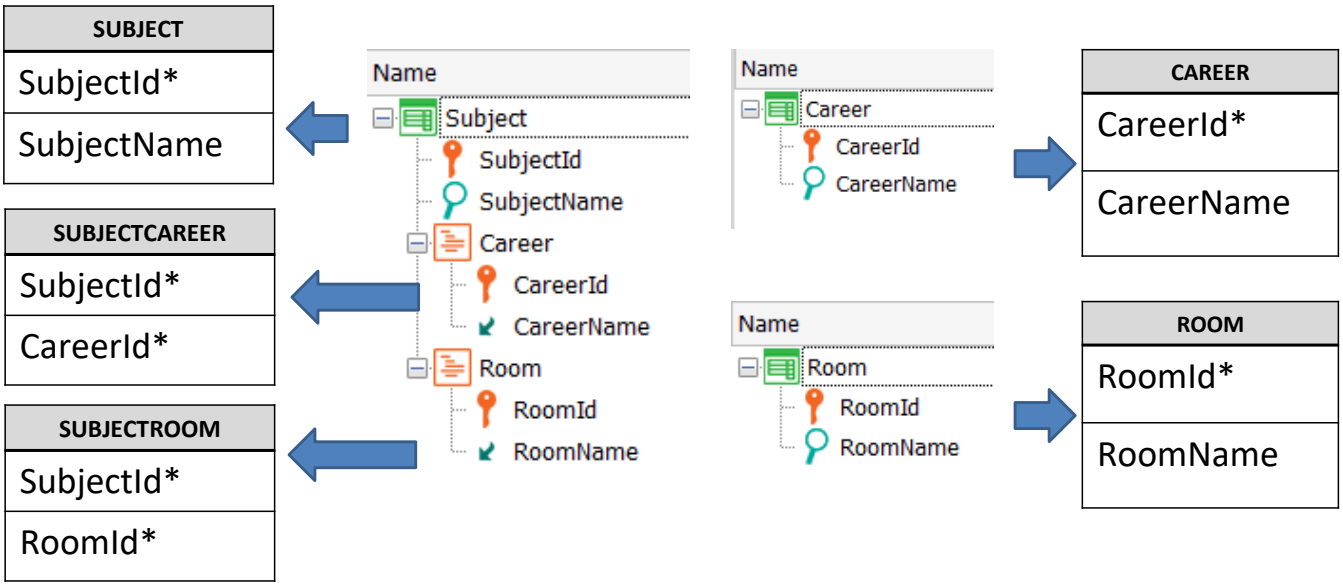
Neste exemplo, a transação Career que possui dois níveis, podemos decompô-la em duas transações de nível único: Career e CareerSubject.

A chave da transação CareerSubject formamos como chave composta com os atributos CareerId e SubjectId.

Observemos que não há uma transação Subject, então SubjectId não será chave estrangeira e SubjectName não poderá ser inferido, será um atributo armazenado na tabela CAREERSUBJECT.

Vemos que com ambos os modelos de transações obtemos exatamente as mesmas tabelas, então podemos afirmar que estes dois desenhos são equivalentes.

Transactions with more than two levels



Uma transação pode conter mais de um subnível. Além disso, cada subnível pode, por sua vez, ter subníveis.

Que uma transação tenha vários subníveis, é o modelo que servirá para representar uma entidade principal (a do cabeçalho) que possui várias entidades com relações 1-N fraca com ela. Ou para representar relações muitos para muitos com várias entidades.

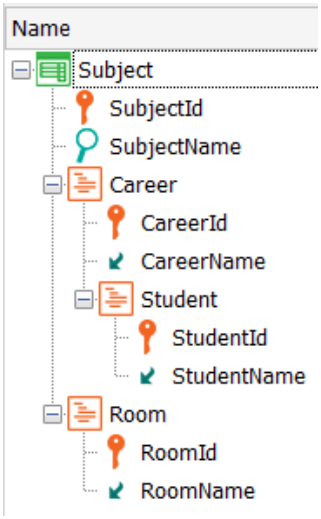
Por exemplo, vamos considerar o caso de que uma matéria, além da relação muitos para muitos com carreira que modelamos antes, pode ser ministrada em várias salas e que cada sala pode ser utilizada para ministrar muitas matérias.

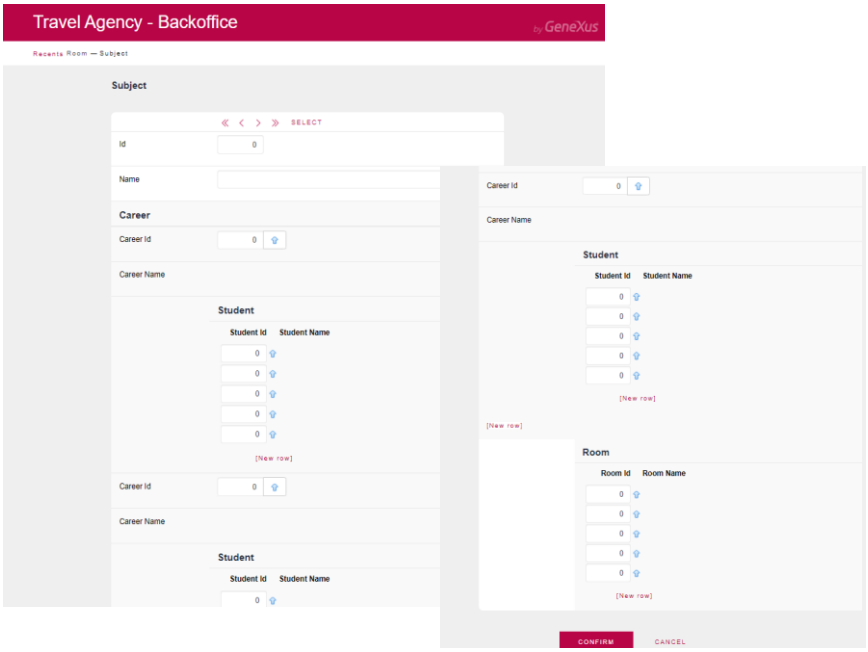
Podemos representar esta realidade criando uma transação Subject com um subnível Career e outro subnível Room, adicionando também ao modelo as transações Career e Room.

Vendo as tabelas que GeneXus construirá, podemos verificar as relações muitos para muitos entre Subject e cada uma das outras entidades.

A tela da transação Subject conterá dois grids, um para cada subnível.

Transactions with nested and sibling levels





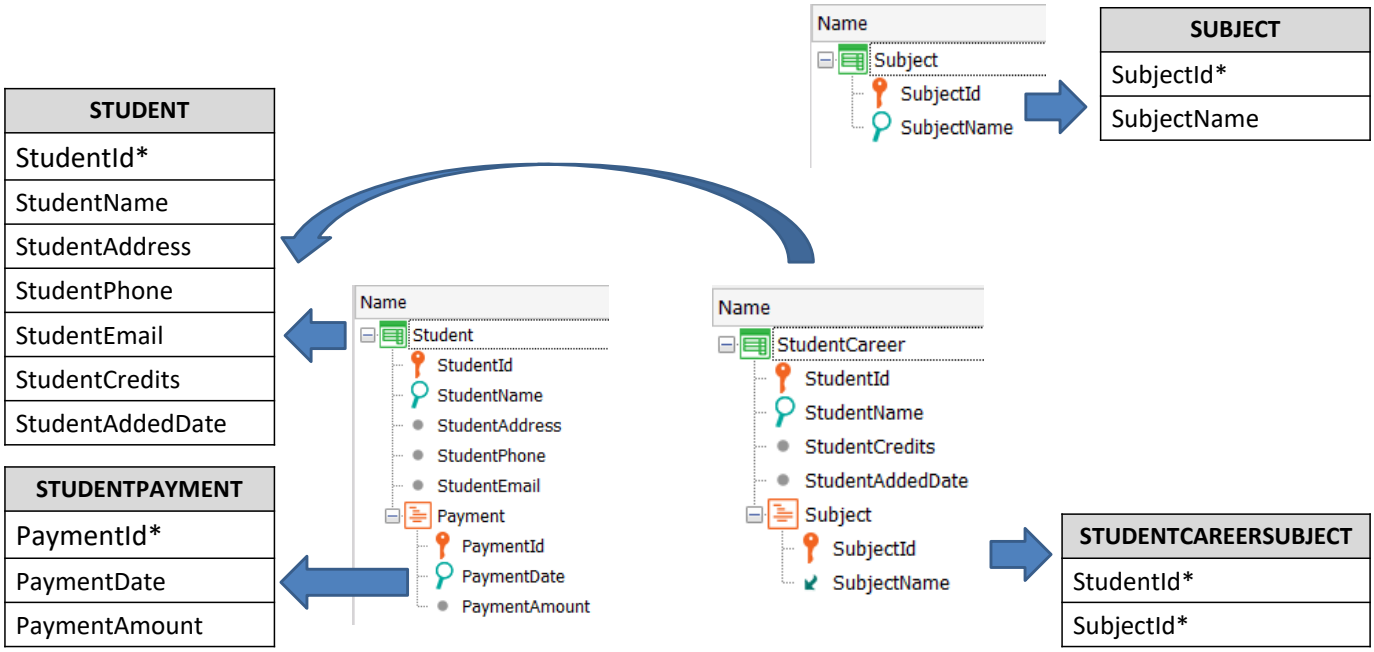
Embora do ponto de vista da modelagem um subnível possa por sua vez conter subníveis, é necessário considerar como ficará a interface de usuário quando for construída a tela da transação.

Neste exemplo, uma matéria tem muitas carreiras e, por sua vez, cada carreira tem inscritos muitos alunos. Além disso, a matéria tem muitas salas onde pode ser ministrada.

Com o desenho que fizemos, aparecem os campos de dados do cabeçalho das matérias, mas depois para cada carreira são adicionados seu cabeçalho e as linhas dos alunos. Esse bloco de dados de cada carreira é repetido várias vezes e no final de tudo está o grid de salas. Este formato causa muito scroll vertical que não é muito confortável para os usuários da aplicação.

O desenvolvedor terá que decidir se em vez de uma transação com estes subníveis não lhe convém aplanar algum nível (como vimos antes) ou se deixa o desenho como está para que sejam geradas as tabelas correspondentes e as telas de entrada de dados são implementadas com web panels ou panels, dependendo da plataforma que vá utilizar.

Parallel transactions with more than one level



Quando queremos ter a informação segmentada de uma mesma entidade, podemos optar por usar transações paralelas.

Estas são transações que possuem o mesmo identificador, mas contém aqueles atributos com informação específica da entidade, conforme queremos segmentá-la.

Por exemplo, se falamos de alunos, poderia ser uma transação com os dados patronímicos e outra com os dados de sua escolaridade, ambas com o mesmo identificador de aluno.

Em particular, pode acontecer que ambas as transações paralelas tenham mais de um nível. Por exemplo, pode acontecer que a transação de dados do aluno tenha um segundo nível com os pagamentos e a transação paralela de dados de escolaridade tenha as matérias cursadas.

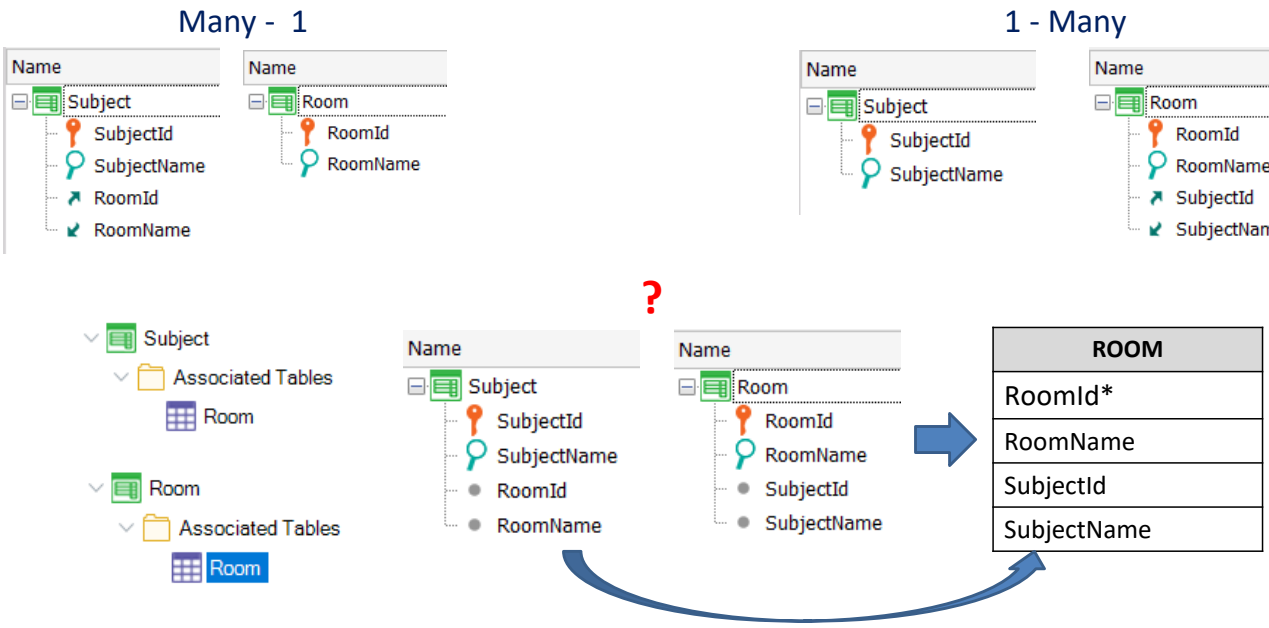
Se analisarmos as tabelas que serão criadas, veremos que a partir da transação Subject será criada a tabela SUBJECT, a partir da transação Student serão criadas duas tabelas: STUDENT e STUDENTPAYMENT e a partir da transação StudentCareer será criada a mesma tabela STUDENT e a tabela STUDENTCAREERSUBJECT.

Se prestarmos atenção à tabela STUDENT, veremos que não apenas estão presentes os atributos do primeiro nível da transação Student, mas também os atributos do primeiro nível da transação StudentCareer. Aqueles atributos que são comuns são adicionados apenas uma vez.

Isso porque o identificador de ambas as transações é StudentId, para o qual GeneXus cria uma única tabela para conter todos os atributos que dependem funcionalmente de StudentId, que neste caso são os atributos de ambas as transações.

Portanto, a partir destas duas transações de dois níveis, não são criadas quatro tabelas, mas apenas três, pois são transações paralelas e com os atributos dos cabeçalhos, será criada uma única tabela que contém todos eles.

Study case: crossed foreign keys



Sabemos que se incluirmos em uma transação um atributo que é chave primária em outra transação, o atributo se tornará chave estrangeira e será estabelecida uma relação 1 para muitos entre ambas as entidades, onde o lado muitos da relação é o da transação onde está a chave estrangeira.

De forma que no exemplo da esquerda, estamos modelando que uma sala pode ser usada para ministrar várias matérias e cada matéria é ministrada em uma única sala (por exemplo, se houver vários turnos de turma). No modelo da direita, uma sala é usada para ministrar uma única matéria, mas a mesma matéria é ministrada em várias salas (por exemplo, se uma matéria requer uma sala com equipamento especial (ex. Laboratório) e existem várias turmas paralelas dessa mesma matéria (em vários Laboratórios).

Agora, o que aconteceria se cruzássemos as chaves primárias de ambas as transações, de forma que uma tenha como chave estrangeira a primária da outra e vice-versa?

Se o implementamos no GeneXus, a primeira coisa que notamos é que ao salvar não são mostradas as setas para cima e para baixo indicando as chaves estrangeiras e atributos inferidos respectivamente.
Vejamos agora as tabelas que serão criadas com este desenho.

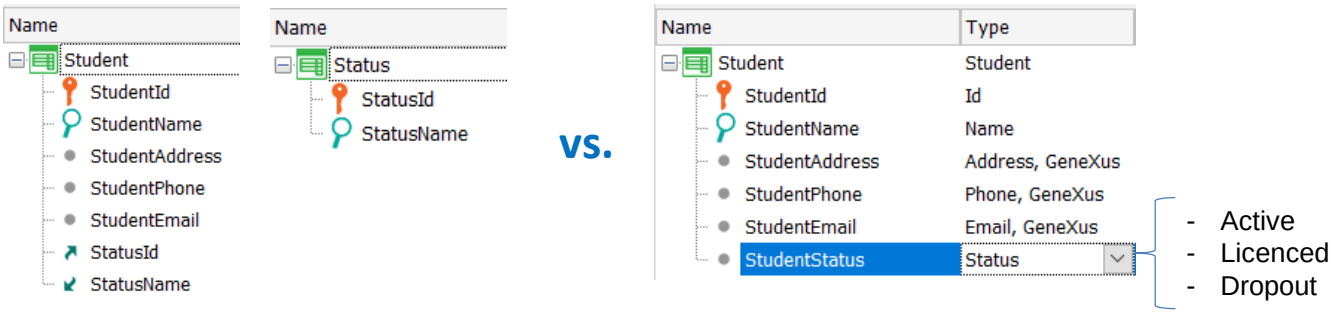
Verificamos que é criada uma única tabela! Neste caso, GeneXus criou a tabela Room, com RoomId como chave primária e o resto dos atributos como atributos secundários. O que aconteceu aqui?
A resposta está nas dependências funcionais dos atributos.

Ao cruzar os identificadores, em uma tabela um atributo é chave primária e, portanto, todos os atributos dessa transação dependem funcionalmente dele, inclusive aquele que é chave estrangeira. Mas na outra transação acontece a mesma coisa, agora a chave primária é outra e o que era chave primária (que agora é estrangeira) depende funcionalmente da chave primária dessa transação. Portanto, GeneXus deve tomar uma decisão já que os dois identificadores não podem ser chaves primárias ao mesmo tempo dado o modelo estabelecido onde existe uma dupla dependência funcional.

Por esse motivo, escolhe apenas uma delas como chave primária e cria uma única tabela com esse identificador, adicionando os demais atributos como secundários, ou seja, dependendo funcionalmente dessa chave primária que escolheu.

Este resultado geralmente não é o esperado pelo que realiza este modelo, que muitas vezes surge por querer implementar uma relação 1 para 1 entre as entidades, ou simplesmente por um erro de modelagem.

Lookup table vs. Enumerated domain



Muitas vezes queremos atribuir um valor a um atributo, selecionando-o em uma lista de valores.

Em geral, para solucionar isto criamos uma transação onde armazenamos a entidade que vamos escolher e os diferentes registros da tabela compõem a lista de valores. Para isto fazemos com que o atributo ao qual queremos atribuir um valor de vários possíveis seja uma chave estrangeira para a transação que contém os valores e eventualmente podemos recuperar outros dados dessa entidade através da tabela estendida.

Um caso típico, continuando com nossos exemplos da universidade, poderia ser selecionar o país de um aluno ou a carreira de uma matéria.

No entanto, se a lista de valores for fixa, ou mudar muito pouco e, em particular, não tiver muitos dados, poderíamos criar um domínio enumerado com esses valores.
Por exemplo, suponhamos que um aluno pode estar em status “ativo” se estiver cursando uma carreira, em status “licenciado” se solicitou um ano sabático, ou em status “fora” se abandonou os cursos.

Nesse caso poderíamos criar um domínio Status com os 3 valores.

No entanto, pode acontecer que mais adiante se decida adicionar um estado novo, por exemplo quando um aluno faz um estágio de estudos em outra universidade e é desejado registrar esse estado que é novo e não coincide com nenhum dos anteriores.

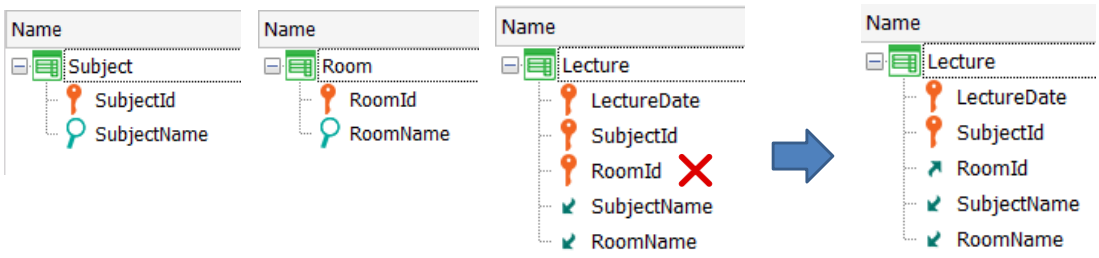
Se usarmos um domínio enumerado e a universidade precisar adicionar esse estado novo, devemos adicionar esse valor ao domínio e depois gerar novamente a aplicação, ou seja, os usuários do sistema na universidade não têm a liberdade para adicionar um dado novo de sua realidade facilmente, ao contrário, eles dependem de que o desenvolvedor volte a gerar os programas, testá-los e colocar em produção a nova versão.

Se, além disso, por algum motivo, fosse utilizado um valor de um domínio enumerado como parte de uma chave primária composta em uma transação, não apenas teria que ser gerada novamente a aplicação, mas também os dados da base de dados, garantindo que não sejam perdidos os dados existentes ao criar uma tabela com uma nova chave primária.

Olhando por esse ponto de vista nunca usaríamos domínios enumerados, no entanto podem existir algumas exceções para valores que sabemos que não podem mudar, como os nomes dos dias da semana ou dos meses do ano.

Em resumo, salvo casos bem conhecidos em que temos certeza de que os valores não mudarão, se tivermos a hipótese, mesmo que seja remota, de que existe a possibilidade de que estes dados podem mudar (como poderia ser os estados de um país, ou os nomes de países existentes), sempre é melhor prática criar uma tabela que contenha esses valores, o que nos garante que usuário da aplicação tenha flexibilidade na hora de atualizar os dados, mesmo que isto implique aumentar as estruturas para manter na base de dados.

Compound primary key vs. simple primary key



SUBJECT

SubjectId	SubjectName
1	Software engineering
2	Basic electronics

ROOM

RoomId	RoomName
1	A101
2	A102
3	A103

LECTURE

LectureDate	SubjectId	RoomId
10/24/2022	1	1
10/24/2022	1	2
10/24/2022	1	3

Quando escolhemos o identificador de uma entidade, por vezes temos várias opções, pois podem existir várias chaves candidatas que podem ser escolhidas como chave primária, seguindo os princípios de unicidade (ou seja, que não pode haver dois registros com a mesma chave) e de irredutibilidade (que a chave deve ser o conjunto mínimo de atributos que identificam os registros de forma única). É bastante comum que seja cumprida a unicidade, mas nem sempre se atenta para a irredutibilidade e acabamos definindo chaves primárias com atributos em excesso.

Vejamos isto com um exemplo.

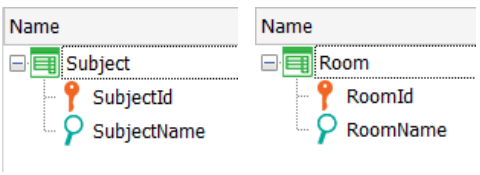
Suponhamos que queremos modelar o caso de uma matéria que é ministrada em uma sala em uma data determinada. Assumimos que essa matéria é ministrada apenas uma vez nessa data. Foram modeladas as transações mostradas, a transação Lecture possui uma chave composta por LectureDate, SubjectId e RoomId, para “garantir” que a combinação da matéria com a data e com a sala seja única.

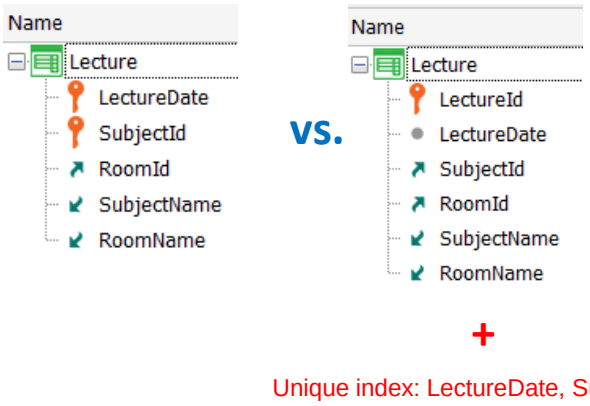
Porém, se analisarmos os dados, veremos que foi possível inserir uma turma para uma matéria em uma data e que seja ministrada em várias salas ao mesmo tempo, pois basta que um dos componentes da chave altere seu valor, para que os demais componentes da chave possam repetir o valor em diferentes registros.

Claramente é desnecessário que o RoomId faça parte da chave, pois somente com o LectureDate e o SubjectId podemos identificar adequadamente a turma para garantir que não se repita a mesma matéria na mesma data. O atributo RoomId

pode estar como chave estrangeira.

Compound primary key vs. simple primary key





Uma dúvida que pode surgir é que em vez de definir uma chave composta em Lecture, formada pelos atributos da data e da matéria cuja combinação queremos que seja única, definimos uma chave artificial LectureId que, por exemplo, seja um identificador numérico autonumerado.

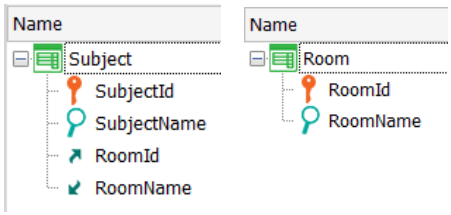
Em ambos os casos, está sendo controlada a unicidade e a chave é irredutível, ou seja, é a mínima para identificar corretamente os registros da tabela.

No entanto, no caso da chave composta, não poderemos modificar os valores da data ou da matéria porque fazem parte da chave e, se quisermos fazê-lo, devemos excluir o registro e definir outro.

Ao contrário, no caso da chave com o identificador da turma, podemos alterar estes dados, embora em contrapartida, ao alterá-los, devemos controlar que não seja repetida a combinação dos valores de data e matéria, pois seria possível definir dois registros com diferentes LectureId, mas o mesmo par data-matéria.

Para evitar isto, poderíamos definir um índice único por data e matéria na tabela LECTURE.

Referential integrity: under the hood



SUBJECT

SubjectId	SubjectName	RoomId
1	Software engineering	3
2	Basic electronics	2
3	Mathematics analysis	1
4	English	4

ROOM

RoomId	RoomName
1	A101
2	A102
3	A103
?	

Um tema em que GeneXus põe especial atenção é de manter uma adequada integridade referencial, de forma a garantir a consistência dos dados. Isto implica que não seja possível a existência de um valor de chave estrangeira que não exista como chave primária na tabela de origem, nem seja possível excluir um valor de uma chave primária que tenha registros relacionados onde essa chave é estrangeira.

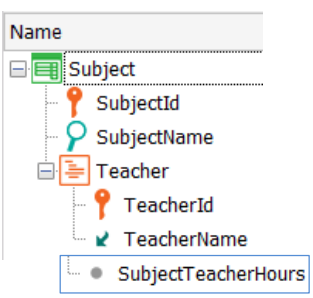
Quando são atualizados dados por meio das transações, seja executando seu form ou um business component dessa transação, o controle de integridade referencial é automático.

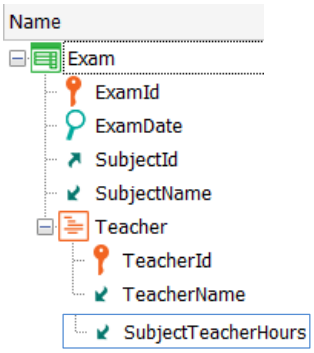
No entanto, existem alguns casos especiais em que pode nos parecer que o controle é realizado, quando na realidade não é cumprido, ou que são realizados controles indesejados e queremos evitá-los, ou que automaticamente é substituído um controle por outro sob certas condições.

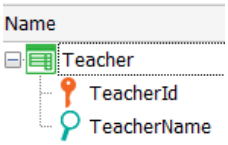
Vamos analisar alguns exemplos que mostram estas situações.

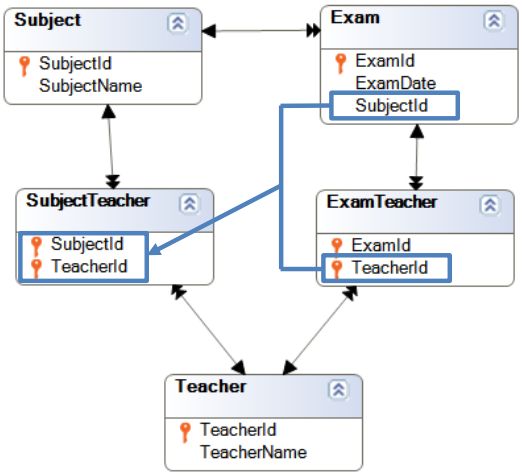
Referential integrity: under the hood

Case 1: Logic subordination









Suponhamos que queremos registrar os exames de uma determinada matéria e o exame será feito por um ou mais professores. Cada matéria tem registrados os professores que a ministram.
Quando adicionamos um exame, deve-se controlar que o professor que se atribua ao exame, ministre a matéria do exame. Isto está garantido pelo desenho?

A resposta é não, pois com este desenho não será realizada automaticamente a verificação. Estamos pretendendo que quando vá inserir um registro na tabela EXAMTEACHER, seja controlado que exista um registro na tabela SUBJECTTEACHER com o valor de SubjectId correspondente ao do exame e com o valor do TeacherId correspondente ao professor que está sendo inserido.

Embora ambos os valores existam em memória, {SubjectId, TeacherId} não formam uma chave estrangeira, pois não estão na mesma tabela.

No entanto, podemos dizer que formam uma chave estrangeira lógica mesmo que isto não exista no nível da base de dados relacional. Como existe uma relação de subordinação lógica entre as tabelas (e não subordinação física), GeneXus não realizará o controle de integridade referencial que necessitamos.

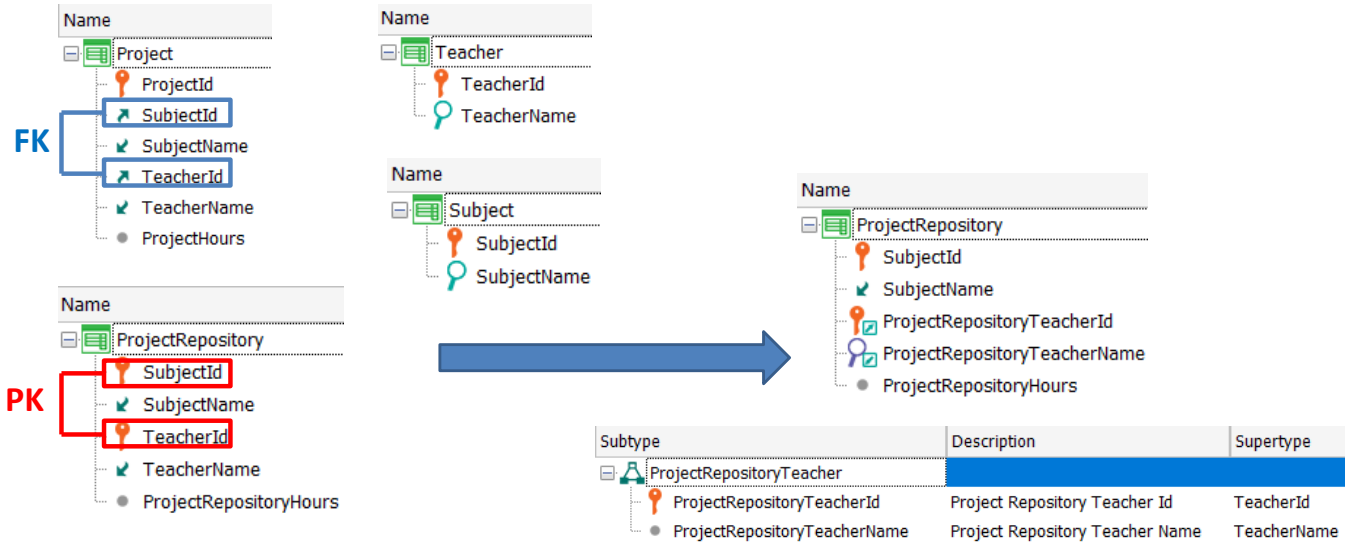
Uma forma de resolver isto é definir uma regra em Exam que invoque um procedimento que realize a verificação e depois com uma regra Error avaliamos o resultado da invocação.

Outra forma de resolver é adicionar no segundo nível da transação Subject um atributo secundário, por exemplo SubjectTeacherHours que registra as horas que

o professor tem atribuídas a essa matéria, de maneira a inferi-lo na transação Exam e assim forçar GeneXus a perceber a relação.

Referential integrity: under the hood

Case 2: Unintended referential integrity checks



Suponhamos agora que na realidade da universidade que estamos desenvolvendo, é desejado registrar os projetos finais que são realizados em determinadas matérias para aprová-la. Um projeto tem um identificador, um professor designado, uma matéria e a quantidade de horas atribuídas a esse projeto.

Além disso, deseja-se ter um repositório com todos os projetos que são realizados na universidade, de modo que ao terminar o semestre os projetos que foram realizados no referido semestre sejam inseridos no repositório.
Para isso, foi criada uma transação ProjectRepository e foi definida uma chave primária composta formada pelo professor e a matéria do projeto.

O problema com este desenho é que SubjectId e TeacherId formam na transação Project uma chave estrangeira para ProjectRepository, de forma que ao tentar inserir um projeto na transação Project, é exigido que o projeto esteja previamente inserido na tabela PROJECTREPOSITORY, o que é impossível, pois primeiro é registrado o projeto e somente após o final do semestre é adicionado ao repositório.

Para evitar isto, podemos criar um grupo de subtipos ProjectRepositoryTeacher, com os subtipos ProjectRepositoryTeacherId subtipo de TeacherId e ProjectRepositoryTeacherName subtipo de TeacherName.
Em seguida, substituímos os atributos TeacherId e TeacherName na transação ProjectRepository pelos subtipos correspondentes.

Desta forma, o que fazemos é com o subtipo, alterar o nome de TeacherId na tabela em que este atributo faz parte da chave primária.

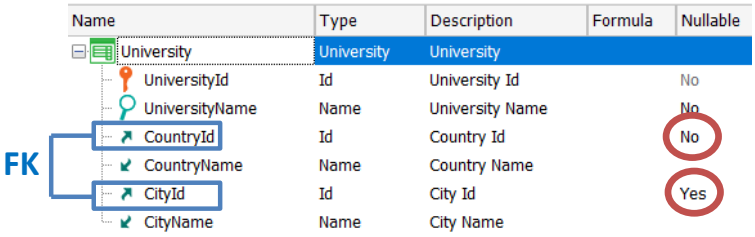
Isto não evita que quando inserimos o identificador de professor na transação ProjectRepository, GeneXus verifique se existe o valor na tabela TEACHER.

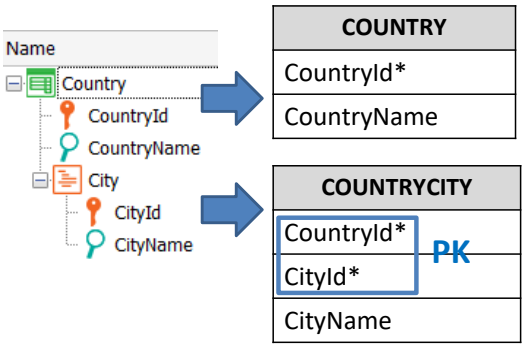
Mas esta mudança de nome evita que o par de atributos {SubjectId, TeacherId} seja identificado como chave estrangeira em Project, pois agora os nomes dos atributos que compõem a chave primária em ProjectRepository são diferentes.

Desta forma, utilizando subtipos, conseguimos evitar que seja realizado um controle de integridade referencial que não era desejado em nossa realidade.

Referential integrity: under the hood

Case 3: Compound foreign key partially nullated





Quando colocamos como nullable um atributo que é chave estrangeira simples, ao inserir um registro se não inserirmos o valor, não será realizado o controle de integridade referencial. Se em vez disso inserirmos um valor, será controlado que o valor inserido na chave estrangeira exista como chave primária na tabela correspondente.

No entanto, quando a chave estrangeira é composta, poderíamos definir como nullable apenas alguns dos atributos da chave.

No exemplo, uma universidade pertence a uma cidade. As cidades são registradas na tabela COUNTRYCITY, que tem como chave primária composta CountryId e CityId. Na transação University, os atributos CountryId e CityId formam uma chave estrangeira composta e, em particular, podemos definir como nullables ambos os atributos, nenhum ou apenas um deles.

Suponhamos que no momento de inserir uma universidade conhecemos o país, mas não a cidade a que pertence, então definimos o atributo CityId com Nullable em Yes.

O que acontecerá quando inserirmos uma universidade? Será feito algum tipo de controle de integridade referencial?
A resposta é sim. Se ao inserir uma universidade não especificarmos o valor de CityId, não será realizado o controle de integridade para verificar se a cidade existe na tabela de cidades, mas como o atributo CountryId ainda possui a propriedade Nullable em No, será realizado um controle de integridade sobre a tabela COUNTRY para verificar se o país inserido existe como país na tabela de países.

Este controle de integridade sobre a tabela COUNTRY não se realizava se ambos os atributos da chave estrangeira composta tivessem o valor Nullable em No, mas o controle se realizava apenas sobre a tabela COUNTRYCITY.

Isso significa que GeneXus adicionou um controle de integridade que originalmente não era feito, já que o atributo CountryId continua sendo uma chave estrangeira em relação à tabela COUNTRY.

Isto demonstra novamente o objetivo de manter sempre os dados consistentes, mesmo nos casos em que são feitas tentativas de desabilitar determinados controles.

Neste resumo, tentamos integrar vários temas relacionadas ao desenho de transações e seu impacto na base de dados e na funcionalidade da aplicação. Convidamos você a aprofundar em alguns destes temas em outros vídeos publicados específicos para cada tema.

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications