M y S Q L:
*Banco de dados relacional
*SGBD open source

JÚLIO AUGUSTO
MACKE PEREIRA

UNICRUZ
Universidade de Cruz Alta
Maio de 2008

1 – Características e desenvolvimento:

Sistema de Gerenciamento de Banco de dados Open Source (código fonte aberto). Programado em linguagem C.

O MySQL É um Banco de Dados Relacional (BDR, baseado em tabelas e os seus relacionamentos). Hoje é muito utilizado e está em crescente desenvolvimento.

Com o uso de tabelas do tipo INNODB, apresenta as características: Atomicidade, Consistência, Isolamento e Durabilidade (ACID).

Apresenta uma grande integração com a linguagem de programação PHP. Sua popularidade, atualmente se deve ao conjunto LAMP (Linux, Apache, MySQL e PHP). Também conta com a facilidade de utilização em IDE's Java: NetBeans 6.0, Java Studio Creator, Eclipse 3.3, etc.

Atualmente com a sua quinta versão incorporou novos recursos, encontrados nos melhores sistemas de banco de dados (stored procedures, two-phase commit, subselects, foreign keys, etc). Uma prova de seu crescimento é que recentemente foi comprado pela Sun (criadora do Java) por US$ 1.000.000.000,00.

A versão 5.0.51a foi lançada em 18 de janeiro de 2008.

1 – Características e desenvolvimento:

O MySQL foi desenvolvido pela TCX em 1996. Foi criado porque seus desenvolvedores precisavam de um banco de dados relacional que pudesse tratar grandes quantidades de dados em máquinas de custo relativamente barato. O MYSQL é um dos bancos de dados relacionais mais rápidos do mercado, apresenta quase todas as funcionalidades dos grandes bancos de dados . MySQL é uma linguagem simples, em que você facilmente pode gravar, alterar e recuperar informações num web site com segurança e rapidez. O MYSQL é executado, principalmente, em sistemas que participam da filosofia UNIX, embora outros sistemas S.O também fornecem suporte, como Windows, por exemplo.

O MYSQL é um sistema de gerenciamento de banco de dados relacional multiencadeado, de código fonte aberto e nível corporativo. O MySQL não é apenas um banco de dados, mas sim um gerenciador de banco de dados.

Página oficial:
http://www.mysql.com/

Página nacional:
http://www.mysqlbrasil.com.br/

Página da Sun:
http://www.sun.com

2 – Programas do MySQL:

Script de inicialização do banco de dados: mysqld_safe
Este arquivo é um script para inicializar o servidor do mysql (mysqld), ele define os parâmetros a serem carregados junto com o servidor.

Servidor do banco de dados: mysqld
O servidor do mysql deve estar rodando para que os clientes possam acessar (consultar, atualizar, inserir ou deletar) os dados e realizar todas as suas tarefas.

Administração do banco de dados: mysqladmin
Um cliente especial para realizar tarefas administrativas (operaçoes com bancos de dados, usuários, ect).

Cliente em modo texto: mysql
Cliente padrão em todas as distribuições do MySQL. É muito usada para tarefas rápidas e para aprender comandos diversos do banco de dados.

A maioria dos programas apresentam uma documentação de ajuda.
Exemplo:
-->> mysqld --help
-->> mysql --help

3 – Algumas empresas que usam o MySQL:

Usado por mais mais de 6 milhões de servidores.
Empresas que usam o MySQL em missões críticas: MP3.com, Motorola, NASA, Silicon Graphics e Texas Instruments.
Empresas que usam em aplicações menores: Nokia, Funcação Bradesco, Google, HP, Sony.
Para se ter uma idéia da quantidade de servidores que utilizam o MySQL, o Wikipedia usa nos seus servidores principais. Dentre os bancos de dados Open Source é o mais usado.

4 – Comandos básicos do cliente do MySQL:

Para efetuar uma conexão com o cliente do MySQL, utilizamos o comando:

-->> mysql

Por padrão, será utilizado o usuário logado no sistema operacional no momento.
Para trocar de usuário, utilizamos o parâmetro "-u [nome_do_usuário]", conforme exemplo:

-->> mysql -u root

Caso o banco esteja exigindo senha para a conexão, utilizamos o parâmetro "-p", exemplo:

-->> mysql -u root -p

Antes de acessarmos o cliente do MySQL, será exigida a senha, após teremos acesso ao banco. Para especificar a porta para conexão, utilizamos "-P [número_da_porta]".
Para acessarmos uma aplicação na inicialização do cliente, usamos o comando "-D [nome_do_banco]", conforme o exemplo:

-->> mysql -u root -p -P 3306 -D information_schema

5 – Dentro do cliente:

O cliente do MySQL nos recebe mostrando a versão do servidor e alguns comandos:

julio@house:~$
julio@house:~$
julio@house:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.0.45-Debian_1ubuntu3.3-log Debian etch distribution
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

A primeira coisa a ser feita agora será criar um banco de dados. Para isso usamos o comando CREATE DATABASE:

mysql> CREATE DATABASE teste1;

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
```

Apartir de agora já temos uma base da dados. Podendo ser utilizada.

5 – Dentro do cliente:

Para visualizarmos as bases de dados cadastradas utilizamos o comando:

mysql> SHOW DATABASES;

Saída gerada:

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| teste1             |
+--------------------+
3 rows in set (0.00 sec)
```

Agora devemos escolher a base de dados para ser usada (no nosso caso, será a teste1):

mysql> use teste1;

Saída gerada:

```
Database changed
```

6 – Dentro de uma aplicação (banco de dados):

Criaremos agora uma tabela para registrar os alunos do nosso curso (com apenas um campo, inicialmente):

mysql> CREATE TABLE alunos (nome varchar(150));

Saída gerada:
Query OK, 0 rows affected (0.01 sec)

Com o comando SHOW podemos mostrar os campos (FIELDS) de nossa tabela:

mysql> SHOW FIELDS FROM alunos:

Saída gerada:

```
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| nome  | varchar(150) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
1 row in set (0.01 sec)
```

6 – Dentro de uma aplicação (banco de dados):

Também com este comando poderemos verificar todas as tabelas de um banco:

mysql> SHOW TABLES FROM teste1;

Saída gerada:

```
+------------------+
| Tables_in_teste1 |
+------------------+
| alunos           |
+------------------+
1 row in set (0.00 sec)
```

7 – Operações básicas do sistema:

O MySQL pode realizar cálculos matemáticos de números inteiros e de fracionários além de trabalhar com strings através do comando SELECT:

mysql> SELECT 3 + 5;

Saída gerada:

```
+-------+
| 3 + 5 |
+-------+
|     8 |
+-------+
1 row in set (0.00 sec)
```

mysql> SELECT 45.5 - 20.4;

Saída gerada:

```
+-------------+
| 45.5 - 20.4 |
+-------------+
|        25.1 |
+-------------+
1 row in set (0.00 sec)
```

7 – Operações básicas do sistema:

mysql> select 45.5 / 20;

Saída gerada:

```
+------------+
| 45.5 / 20  |
+------------+
|    2.27500 |
+------------+
1 row in set (0.03 sec)
```

mysql> SELECT SQRT(16);

Saída gerada:

```
+-----------+
| SQRT(16)  |
+-----------+
|         4 |
+-----------+
1 row in set (0.00 sec)
```

7 – Operações básicas do sistema:

mysql> SELECT pow(9, 2);

Saída gerada:

```
+-----------+
| pow(9, 2) |
+-----------+
|        81 |
+-----------+
1 row in set (0.00 sec)
```

mysql> SELECT pow(5, 2);

Saída gerada:

```
+-----------+
| pow(5, 2) |
+-----------+
|        25 |
+-----------+
1 row in set (0.00 sec)
```

mysql> SELECT pow(2, 4);

Saída gerada:

```
+-----------+
| pow(2, 4) |
+-----------+
|        16 |
+-----------+
1 row in set (0.00 sec)
```

7 – Operações básicas do sistema:

Também efetua concatenações de strings:

mysql> SELECT 'JAT';

Saída gerada:

```
+-----+
| JAT |
+-----+
| JAT |
+-----+
1 row in set (0.00 sec)
```

mysql> SELECT 'JAT' ' - JORNADA DE ATUALIZACAO TECNOLOGICA';

Saída gerada:

```
+-----------------------------------------+
| JAT                                     |
+-----------------------------------------+
| JAT - JORNADA DE ATUALIZACAO TECNOLOGICA |
+-----------------------------------------+
1 row in set (0.00 sec)
```

Obs: Não utilizamos o sinal de mais (+), pois o MySQL tentaria somar os valores. Como não se tratam de valores matemáticos, estes não pode ser somados. Adiante aprenderemos como concatenar strings com resultados de consultas.

8 – Consulta ao banco:

Mostrando valores de uma tabela:
O processo para obtermos os valores de uma tabela, é relativamente simples. Usamos o comando SELECT (seleção), seguido das colunas que queremos visualizar (ou asterísco para todas), a palavra reservada FROM (de) e após o nome da tabela. Como no exemplo:

mysql> SELECT * FROM alunos;

Saída gerada:

```
Empty set (0.01 sec)
```

Obs: Como esta tabela não foi povoada, nenhum aluno foi retornado.

9 – Inserções em tabelas:

O processo de inserção consiste em especificarmos em quais colunas desejamos inserir e quais os dados. Como nossa tabela apresenta apenas uma coluna, este processo é simples:

mysql> INSERT INTO alunos (nome) VALUES ('Joazinho');

Saída gerada:

```
Query OK, 1 row affected (0.03 sec)
```

O comando INSERT (insere) grava os dados. INTO (em) serve para especificarmos a tabela a receber os dados. Escrevemos entre parênteses a(s) coluna(s), após especificamos em VALUES (valores) os dados. Os dados a serem gravados ficam dentro do segundo parênteses. Caso mais de uma coluna seja gravada, os dados deverão ser separados por vírgulas (Mostrado mais adiante).

Inserções múltiplas: No próximo comando, serão gravados dois registros.

mysql> INSERT INTO alunos (nome) VALUES ('Mariazinha'), ('Juquinha');

Saída gerada:

```
Query OK, 2 rows affected (0.03 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

9 – Inserções em tabelas:

Agora poderemos consultar os dados inseridos na tabela alunos:

mysql> SELECT * FROM alunos;

Saída gerada:

```
+-------------+
| nome        |
+-------------+
| Joazinho    |
| Mariazinha  |
| Juquinha    |
+-------------+
3 rows in set (0.0 sec)
```

10 – Remoção de dados:

Durante a remoção de dados devemos descrever a tabela e uma condição (por exemplo, remover pessoa com determinado nome).
Veja o exemplo:

DELETE FROM alunos WHERE nome = 'Juquinha';

DELETE FROM (delete de) diz que devemos remover da tabela alunos algum (ou alguns) dado (s). A clausula WHERE (onde) limita os dados a serem apagados, no caso, apenas onde o nome é igual a Juquinha: WHERE (onde) nome = 'Juquinha'.

mysql> DELETE FROM alunos WHERE nome = 'Juquinha';

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
```

Para remover todos os dados de uma tabela, basta não especificarmos uma clausula. Muito cuidado ao utilizar o DELETE. Pois podemos remover dados acidentalmente. Removendo todos os dados inseridos em uma tabela:

mysql> DELETE FROM alunos;

Saída gerada:

```
Query OK, 2 rows affected (0.03 sec)
```

10 – Remoção de dados:

Como já tínhamos removido uma pessoa da tabela, agora apenas duas pessoas foram removidas. Nossa tabela se encontra vazia novamente.

mysql> SELECT * FROM alunos;

Saída gerada:

```
Empty set (0.00 sec)
```

11 – Tipos de dados básicos:

Tabelas em MySQL podem utilizar vários tipos de variáveis. Aqui segue uma amostra de alguns:

TIPOS NUMÉRICOS:

TINYINT -->> Um byte. São oito bits. Com uma variação de oito bits temos dois elevado na oito. Isto dá 256 números. Com valores somente positivos, temos de 0 até 255. Com sinal temos de -127 até 128. Utilizado para pequenos números.

INT -->> Quatro bytes. Temos 32 bits. Conseguimos uma variação de dois elevado na trinta e dois (4.294.967.296). Sem sinal (positivos) temos de 0 até 4.294.967.295. Com sinal a representação fica entre -2.147.483.648 até 2.147.483.647.

BIGINT -->> Oito bytes. Armazena valores extremamente grandes.

FLOAT -->> Quatro bytes. Variáveis com vírgula. Representam valores fracionários.

DOUBLE -->> Oito bytes. Idênticos ao FLOAT, com a diferença de ter o dobro do tamanho de armazenamento.

Obs: Nos tipos numéricos, o padrão de criação de tabela é com valores positivos e negativos (com sinal). Para campos somente positivos (sem sinal), devemos especificar na criação do campo que este não terá sinal (UNSIGNED).
Em ambos os casos, durante a inserção de um número que ultrapasse o limite, será resgistrado no campo esse limite. O mesmo se aplica para valores negativos, o limite inferior será registrado (ou zero, caso este campo não tenha sinal).

11 – Tipos de dados básicos:

TIPOS DE CARACTERES:

CHAR -->> Um byte. Podemos representar até 256 caracteres.

VARCHAR -->> Cadeia de caracteres. Cada caracter ocupa um byte. Este tipo de dado acrescenta um byte ao final da sequência. Este byte final é usado para especificar a forma de ordenação da cadeia de caracteres (se maiúsculas e minúsculas serão consideradas).

TEXT -->> Cadeia de caracteres com espaço para 65535 caracteres.

MEDIUMTEXT -->> Idem ao TEXT, mas com armazenamento maior. Podemos armazenar até 16777215 caracteres.

TIPOS DE DATAS:

Date: -->> Armazena uma data. A margem de valores vai desde o 1 de Janeiro de 1001 ao 31 de dezembro de 9999. O formato de armazenamento é de ano-mes-dia (2008-05-16).

DateTime: Combinação de data e hora. A margem de valores vai desde o 1 ed Janeiro de 1001 às 0 horas, 0 minutos e 0 segundos ao 31 de Dezembro de 9999 às 23 horas, 59 minutos e 59 segundos. O formato de armazenamento é de ano-mes-dia horas:minutos:segundos (2008-05-16 20:30:00).

12 – Chaves primárias:

São usadas para identificar registros em uma tabela.

As chaves primárias são utilizadas para prover aos dados inseridos uma garantia de que não haverão repetições. Um campo que é setado como chave primária nunca terá valor repetido na mesma tabela. Chaves primárias também são utilizadas para relacionarmos diversas tabelas (isto será explicado mais adiante).

Um exemplo simples: Ao efetuarmos a primeira remoção da tabela alunos, removemos o aluno 'Juquinha'. Mas se tivéssemos dois alunos com este nome, os dois seriam removidos. (Query OK, 2 rows affected). A linha desejada seria removida, mas outra neste caso, também seria deletada. Se tivéssemos uma chave primária poderíamos modificar a remoção para que somente, o aluno com chave primária N fosse removido.

As chaves primárias normalmente são valores inteiros. Como são únicas, não podem ser omitidas (são portanto, não nulas) e positivas. As chaves podem ser outros tipos de dados e também podem ser compostas (dois ou mais campos formam uma chave), de maneira que os campos não podem ter a mesma combinação.

Criando uma tabela com chave primária:

```
mysql> CREATE TABLE cursos (
    -> id int primary key auto_increment,
    -> nome varchar(150),
    -> descricao varchar(150),
    -> obs varchar(150)
    -> );
```

Saída gerada:

```
Query OK, 0 rows affected (0.06 sec)
```

12 – Chaves primárias:

Agora temos uma tabela com quatro campos. Durante a criação da tabela, separamos com vírgula os campos (colunas). Visualizando os campos (colunas) da tabela, temos:

mysql> SHOW FIELDS FROM cursos;

Saída gerada:

```
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int(11)      | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(150) | YES  |     | NULL    |                |
| descricao | varchar(150) | YES  |     | NULL    |                |
| obs       | varchar(150) | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
4 rows in set (0.04 sec)
```

Analisando, percebemos que a coluna id está marcada como chave primária (Key - PRI) e também está marcada como não nula (Null - NO). Agora temos a certeza que não haverá repetições de identificação de curso e também que todos os cursos criados terão uma id. No MySQL a inserção de chave primária pode ser feita de maneira automática pelo servidor. Vamos verificar a funcionalidade da chave primária: O primeiro curso criado será o de MySQL:

mysql> INSERT INTO cursos (id, nome, descricao) VALUES (1, 'MySQL', 'Curso Pratico de MySQL');

Saída gerada:

```
Query OK, 1 row affected (0.02 sec)
```

12 – Chaves primárias:

Agora o curso de Java, com a mesma id (o que seria um erro):

mysql> INSERT INTO cursos (id, nome, descricao) VALUES (1, 'Java', 'Curso de Linguagem de programacao');

Saída gerada:

```
ERROR 1062 (23000): Duplicate entry '1' for key 1
```

Agora o curso de Java, mas de maneira correta:

mysql> INSERT INTO cursos (id, nome, descricao) VALUES (2, 'Java', 'Curso de Linguagem de programacao');

Saída gerada:

```
Query OK, 1 row affected (0.00 sec)
```

Agora todos os cursos:

mysql> INSERT INTO cursos (nome, descricao) VALUES ('Manutencao', 'Manutencao de Computadores'), ('PHP', 'Linguagem de programacao');

Saída gerada:

```
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

mysql> INSERT INTO cursos (nome, descricao, obs) VALUES ('J2ME', 'Java Micro Edition', 'Dois professores');

Saída gerada:

```
Query OK, 1 row affected (0.00 sec)
```

12 – Chaves primárias:

Mostrando todos os dados da tabela cursos:

mysql> SELECT * FROM cursos;

Saída gerada:

```
+----+------------+-------------------------------------+------------------+
| id | nome       | descricao                           | obs              |
+----+------------+-------------------------------------+------------------+
|  1 | MySQL      | Curso Pratico de MySQL              | NULL             |
|  2 | Java       | Curso de Linguagem de programacao   | NULL             |
|  3 | Manutencao | Manutencao de Computadores          | NULL             |
|  4 | PHP        | Linguagem de programacao            | NULL             |
|  5 | J2ME       | Java Micro Edition                  | Dois professores |
+----+------------+-------------------------------------+------------------+
5 rows in set (0.00 sec)
```

Agora temos uma maneira mais simples de remover um registro. Podemos remover baseados em sua id: Desta forma:

mysql> DELETE FROM cursos WHERE id = 5;

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
```

Mas como este curso está correto, vamos reinseri-lo:

mysql> INSERT INTO cursos (nome, descricao, obs) VALUES ('J2ME', 'Java Micro Edition', 'Dois professores');

Saída gerada:

```
Query OK, 1 row affected (0.00 sec)
```

12 – Chaves primárias:

Observe que sua id é diferente da outra. O curso com id igual a cinco foi removido. Este curso é uma nova inserção no banco, portanto tem uma nova identidade.

mysql> SELECT * FROM cursos;

Saída gerada:

```
+----+------------+-------------------------------------+------------------+
| id | nome       | descricao                           | obs              |
+----+------------+-------------------------------------+------------------+
|  1 | MySQL      | Curso Pratico de MySQL              | NULL             |
|  2 | Java       | Curso de Linguagem de programacao   | NULL             |
|  3 | Manutencao | Manutencao de Computadores          | NULL             |
|  4 | PHP        | Linguagem de programacao            | NULL             |
|  6 | J2ME       | Java Micro Edition                  | Dois professores |
+----+------------+-------------------------------------+------------------+
5 rows in set (0.00 sec)
```

Assim como as remoções do banco ficam mais fáceis com o uso de chaves primárias, as alterações (UPDATE) também ficam mais simplificadas. Vamos padronizar as descrições da tabela. Este comando será explicado mais adiante.

mysql> UPDATE cursos SET descricao = 'Curso de Linguagem de Programacao' WHERE (id = 2 OR id = 4);

Saída gerada:

```
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

Este comando atualiza a tabela cursos. Setando (SET) descricao como 'Curso de Linguagem de Programacao' onde (WHERE) id for 2 ou 4.
Atualizamos dois cursos. A descrição ficou igual para os cursos 2 e 4. As duas linhas foram alteradas.

12 – Chaves primárias:

mysql> SELECT * FROM cursos;

Saída gerada:

```
+----+------------+-------------------------------------+------------------+
| id | nome       | descricao                           | obs              |
+----+------------+-------------------------------------+------------------+
|  1 | MySQL      | Curso Pratico de MySQL              | NULL             |
|  2 | Java       | Curso de Linguagem de Programacao   | NULL             |
|  3 | Manutencao | Manutencao de Computadores          | NULL             |
|  4 | PHP        | Curso de Linguagem de Programacao   | NULL             |
|  6 | J2ME       | Java Micro Edition                  | Dois professores |
+----+------------+-------------------------------------+------------------+
5 rows in set (0.00 sec)
```

Como podemos perceber, as chaves primárias são de extrema importância e devem (preferencialmente) estar presentes em todas as tabelas criadas.

13 – Chaves estrangeiras:

Este recurso é muito importante em banco de dados relacionais. O MySQL fornece todos os recursos necessários para lidarmos com as chaves estrangeiras. Estas chaves são usadas para relacionarmos tabelas entre si. Analise o seguinte caso: Nossa tabela cursos tem informarções sobre os cursos fornecidos na Jornada de Atualização Tecnológica. Podemos melhorar nossa tabela de alunos para que tenhamos mais informações nessa tabela. Como ainda não estudamos alterações em tabelas (comando ALTER TABLE), vamos deletar a tabela alunos (comando DROP TABLE):

```
mysql> DROP TABLE alunos;
```

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

Inicialmente poderemos colocar uma chave primária e uma coluna para observações diversas. Nosso comando ficaria:

```
CREATE TABLE alunos (
        id int primary key auto_increment,
        nome varchar(150),
        obs varchar(150)
);
```

Durante a jornada, o aluno poderá fazer somente um curso. Então cada aluno estará ligado a apenas um curso da tabelas cursos. A tabela alunos poderá informar qual curso o aluno faz. Essa informação não poderia ser gravada na tabela cursos, pois não sabemos quanto cada curso tem de alunos. Criaremos um campo (coluna) na tabela alunos, informando qual curso o aluno faz. Vamos chamar essa coluna de cod_curso. Nela, guardaremos apenas o código (id) da tabela cursos. Exemplo: Um aluno que tenha na coluna cod_curso o valor 4, faz o curso de PHP. Um aluno com o código 1, faz o curso de MySQL.

13 – Chaves estrangeiras:

Para mantermos a integridade do nosso sistema, todo aluno deverá participar de algum curso (e apenas um). E este curso deverá estar cadastrado na tabela de cursos. Um aluno não poderá ter o cod_curso igual a 8 por exemplo. Este curso não existe, portanto uma tentativa de inserção na tabela alunos com cod_curso igual a 8, deverá resultar em um erro. Agora veja como pode ficar nossa tabela:

```
CREATE TABLE alunos (
        id int primary key auto_increment,
        nome varchar(150),
        obs varchar(150),
        cod_curso int
);
```

Como a chave primária da tabela cursos (id) é uma variável inteira, nosso cod_curso também será. Para que de fato tenhamos nossa chave estrangeira, devemos inserir uma constraint informando qual campo é chave estrangeira e a qual tabela (e campo da tabela) ela está se referenciando:

```
CREATE TABLE alunos (
        id int primary key auto_increment,
        nome varchar(150),
        obs varchar(150),
        cod_curso int,
        constraint chave_ext_curso_aluno foreign key (cod_curso) references cursos(id)
);
```

13 – Chaves estrangeiras:

Nossa última coluna é a chave estrangeira. A palavra constraint indica que teremos uma chave, chave_ext_curso_aluno é o seu nome (assim como temos os outros nomes: id, nome, obs, cod_curso), foreign key indica que temos uma chave estrangeira. Entre parênteses temos a coluna nesta tabela que é a chave. Para fazermos a referência à tabela cursos, usamos a palavra references seguida do nome da tabela e entre parênteses a coluna. Agora podemos executar no cliente do MySQL a criação da tabela:

```
mysql> CREATE TABLE alunos (
    -> id int primary key auto_increment,
    -> nome varchar(150),
    -> obs varchar(150),
    -> cod_curso int,
    -> CONSTRAINT chave_ext_curso_aluno FOREIGN KEY (cod_curso) REFERENCES cursos(id)
    -> );
```

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

13 – Chaves estrangeiras:

Com o comando SHOW vimos anteriormente as colunas de uma tabela, executando agora teremos:

mysql> SHOW FIELDS FROM alunos;

Saída gerada:

```
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int(11)      | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(150) | YES  |     | NULL    |                |
| obs       | varchar(150) | YES  |     | NULL    |                |
| cod_curso | int(11)      | YES  | MUL | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

Mas ainda não vemos o nome de nossa chave. Para mais detalhes sobre chaves, visualizamos todos os índices de uma tabela:

mysql> SHOW INDEXES FROM alunos;

Saída gerada:

```
+--------+------------+----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
| Table  | Non_unique | Key_name             | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+--------+------------+----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
| alunos |          0 | PRIMARY              |            1 | id          | A         |           0 |     NULL | NULL   |      | BTREE      |         |
| alunos |          1 | chave_ext_curso_aluno |            1 | cod_curso   | A         |        NULL |     NULL | NULL   | YES  | BTREE      |         |
+--------+------------+----------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
2 rows in set (0.00 sec)
```

13 – Chaves estrangeiras:

Agora temos a tabela aluno com a sua chave primária e também com uma chave estrangeira, fazendo referência a tabela cursos. Algumas considerações sobre chaves estrangeiras:

* Só poderemos inserir um aluno se o seu cod_curso fizer referência a um curso existente (cursos.id = alunos.cod_curso).
* Não poderemos inserir um aluno se a tabela de cursos estiver vazia (pois não haverá um curso para o aluno estar ligado).
* Alterações em alunos.cod_curso só poderão ser feitas se o cod_curso tiver relação com um outro curso existente (cursos.id).
* Um curso só poderá ser removido, se nenhum aluno estiver cadastrado nele (alunos.cod_curso precisa de um valor igual em cursos.id).
* Uma alternativa para a remoção de um curso com alunos ligados a ele, seria usar o comando DROP TABLE com o recurso CASCATE.
* Outra alternativa, seria baixar as chaves (DISABLE KEYS).
* Integridade referencial em MySQL só é possível com o tipo de tabelas INNODB. Usaremos em seguida.

Vamos inserir algumas pessoas dentro da tabela alunos (e vincula-las a um curso).

mysql> INSERT INTO alunos (nome, cod_curso) VALUES ('Juquinha', 1), ('Pedrinho', 2), ('Mariazinha', 3), ('Joao', 3);

Saída gerada:

```
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

13 – Chaves estrangeiras:

Temos duas pessoas fazendo o curso de Manutenção, uma fazendo o curso de Java e outra fazendo o de MySQL.

O MySQL suporta diversos tipos de tabelas (InnoDB, MyISAM, Heap, Merge, BDB, Memory). Mas apenas o InnoDB apresenta integridade referencial. Por padrão todas as tabelas criadas no MySQL são do tipo MyISAM. Como este tipo não apresenta integridade, as chaves estrangeiras servem apenas para documentação. Vamos fazer um teste:

mysql> INSERT INTO alunos (nome, cod_curso) VALUES ('Leia',89);

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
```

Não existe em nosso sistema um curso com o código 89. Inserimos uma aluna que não está matriculada em um curso correto. O controle de integridade pode ser feito pelo sistema no qual o MySQL está rodando (exemplo, uma aplicação em Java ou um site em PHP). Mas como aqui estamos estudando a integridade referencial, podemos mudar o tipo de tabela para InnoDB:

mysql> UPDATE alunos SET cod_curso = 4 WHERE id = 5;

Saída gerada:

```
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

mysql> ALTER TABLE alunos TYPE=innodb;

Saída gerada:

```
Query OK, 5 rows affected, 1 warning (0.20 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

13 – Chaves estrangeiras:

mysql> ALTER TABLE cursos TYPE=innodb;

Saída gerada:

```
Query OK, 5 rows affected, 1 warning (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Agora nossa aplicação conta com tabelas do tipo InnoDB. Façamos o teste:

mysql> INSERT INTO alunos (nome, cod_curso) VALUES ('Aninha',9);

Saída gerada:

```
Query OK, 1 row affected (0.10 sec)
```

Como podemos ver, o servidor deixou criarmos uma aluna que está fora de um curso válido. Para corrigirmos essa falha devemos reconstruir nossas tabelas:

mysql> DROP TABLE cursos;

Saída gerada:

```
Query OK, 0 rows affected (1.94 sec)
```

mysql> DROP TABLE alunos;

Saída gerada:

```
Query OK, 0 rows affected (0.18 sec)
```

13 – Chaves estrangeiras:

```
mysql> CREATE TABLE cursos (
    -> id int primary key auto_increment,
    -> nome varchar(150),
    -> descricao varchar(150),
    -> obs varchar(150)
    -> ) ENGINE = INNODB;
```

Saída gerada:

```
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> CREATE TABLE alunos (
    -> id int primary key auto_increment,
    -> nome varchar(150),
    -> obs varchar(150),
    -> cod_curso int,
    -> CONSTRAINT chave_ext_curso_aluno FOREIGN KEY (cod_curso) REFERENCES cursos(id)
    -> ) ENGINE = INNODB;
```

Saída gerada:

```
Query OK, 0 rows affected (0.07 sec)
```

```
mysql> INSERT INTO cursos (nome, descricao) VALUES ('MySQL', 'Curso Pratico de MySQL');
```

Saída gerada:

```
Query OK, 1 row affected (0.13 sec)
```

13 – Chaves estrangeiras:

mysql> INSERT INTO cursos (nome, descricao) VALUES ('Java', 'Curso de Linguagem de Programacao');

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
```

mysql> INSERT INTO cursos (nome, descricao) VALUES ('Manutencao', 'Manutencao de Computadores'), ('PHP', 'Curso de Linguagem de Programacao');

Saída gerada:

```
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

mysql> INSERT INTO cursos (nome, descricao, obs) VALUES ('J2ME', 'Java Micro Edition', 'Dois professores');

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
```

mysql> INSERT INTO alunos (nome, cod_curso) VALUES ('Juquinha', 1), ('Pedrinho', 2), ('Mariazinha', 3), ('Joao', 3);

Saída gerada:

```
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

mysql> INSERT INTO alunos (nome, cod_curso) VALUES ('Leia', 5);

Saída gerada:

```
Query OK, 1 row affected (0.00 sec)
```

13 – Chaves estrangeiras:

Agora nossa aplicação conta realmente com integridade referencial:

mysql> INSERT INTO alunos (nome, cod_curso) VALUES ('Adao', 54);

Saída gerada:

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
  (`teste1/alunos`,  CONSTRAINT  `chave_ext_curso_aluno`  FOREIGN  KEY  (`cod_curso`)
  REFERENCES `cursos` (`id`))
```

Outros exemplos de chaves primárias e estrangeiras:

```
CREATE DATABASE universidade;
USE universidade
CREATE TABLE Cursos ( id int primary key auto_increment, descricao varchar(200), obs varchar(200)) ENGINE =
   INNODB;
CREATE TABLE Disciplinas ( id int primary key auto_increment, descricao varchar(200), obs varchar(200), cod_curso
   int, constraint fk_curso_disciplina foreign key (cod_curso) references Cursos(id)) ENGINE = INNODB;

CREATE DATABASE discoteca;
USE discoteca
CREATE TABLE Albuns ( id int primary key auto_increment, nome_album varchar(200), artista varchar(200), obs
   varchar(200), ano date) ENGINE =INNODB;
CREATE TABLE Musicas ( id int primary key auto_increment, nome_musica varchar(200), obs varchar(200),
   cod_album int, constraint fk_album_musica foreign key (cod_album) references Albuns(id)) ENGINE = INNODB;
```

14 – Seleção, cross join, inner join e subseleção:

CROSS JOINS:

Agora já podemos efetuar seleções entre duas tabelas. A maneira mais simples seria:

mysql> SELECT * FROM alunos, cursos;

Saída gerada:

```
+----+------------+------+-----------+----+------------+-----------------------------------+------------------+
| id | nome       | obs  | cod_curso | id | nome       | descricao                         | obs              |
+----+------------+------+-----------+----+------------+-----------------------------------+------------------+
|  1 | Juquinha   | NULL |         1 |  1 | MySQL      | Curso Pratico de MySQL            | NULL             |
|  2 | Pedrinho   | NULL |         2 |  1 | MySQL      | Curso Pratico de MySQL            | NULL             |
|  3 | Mariazinha | NULL |         3 |  1 | MySQL      | Curso Pratico de MySQL            | NULL             |
|  4 | Joao       | NULL |         3 |  1 | MySQL      | Curso Pratico de MySQL            | NULL             |
|  5 | Leia       | NULL |         5 |  1 | MySQL      | Curso Pratico de MySQL            | NULL             |
|  1 | Juquinha   | NULL |         1 |  2 | Java       | Curso de Linguagem de Programacao | NULL             |
|  2 | Pedrinho   | NULL |         2 |  2 | Java       | Curso de Linguagem de Programacao | NULL             |
|  3 | Mariazinha | NULL |         3 |  2 | Java       | Curso de Linguagem de Programacao | NULL             |
|  4 | Joao       | NULL |         3 |  2 | Java       | Curso de Linguagem de Programacao | NULL             |
|  5 | Leia       | NULL |         5 |  2 | Java       | Curso de Linguagem de Programacao | NULL             |
|  1 | Juquinha   | NULL |         1 |  3 | Manutencao | Manutencao de Computadores        | NULL             |
|  2 | Pedrinho   | NULL |         2 |  3 | Manutencao | Manutencao de Computadores        | NULL             |
|  3 | Mariazinha | NULL |         3 |  3 | Manutencao | Manutencao de Computadores        | NULL             |
|  4 | Joao       | NULL |         3 |  3 | Manutencao | Manutencao de Computadores        | NULL             |
|  5 | Leia       | NULL |         5 |  3 | Manutencao | Manutencao de Computadores        | NULL             |
|  1 | Juquinha   | NULL |         1 |  4 | PHP        | Curso de Linguagem de Programacao | NULL             |
|  2 | Pedrinho   | NULL |         2 |  4 | PHP        | Curso de Linguagem de Programacao | NULL             |
|  3 | Mariazinha | NULL |         3 |  4 | PHP        | Curso de Linguagem de Programacao | NULL             |
|  4 | Joao       | NULL |         3 |  4 | PHP        | Curso de Linguagem de Programacao | NULL             |
|  5 | Leia       | NULL |         5 |  4 | PHP        | Curso de Linguagem de Programacao | NULL             |
|  1 | Juquinha   | NULL |         1 |  5 | J2ME       | Java Micro Edition                | Dois professores |
|  2 | Pedrinho   | NULL |         2 |  5 | J2ME       | Java Micro Edition                | Dois professores |
|  3 | Mariazinha | NULL |         3 |  5 | J2ME       | Java Micro Edition                | Dois professores |
|  4 | Joao       | NULL |         3 |  5 | J2ME       | Java Micro Edition                | Dois professores |
|  5 | Leia       | NULL |         5 |  5 | J2ME       | Java Micro Edition                | Dois professores |
+----+------------+------+-----------+----+------------+-----------------------------------+------------------+
25 rows in set (0.01 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Uma seleção entre duas tabelas faz simplesmente a multiplicação entre as elas, combinando todas as linhas de uma tabela com todas as linhas da outra tabela. Temos cinco cursos cadastrados. Cada curso será mostrado juntamente com todos os alunos (cinco alunos). Uma seleção dessa maneira é chamada de CROSS JOIN. Temos duas tabelas com cinco registros cada, logo o número de registros resultantes será 25 (5 X 5). Se inserirmos uma terceira tabela na consulta, teremos a combinação de todas por todas.

INNER JOINS:

Para nossa aplicação, está consulta não é interessante, pois devemos saber em qual curso um aluno está matriculado. E não a combinação de cada um dos alunos com cada um dos cursos. Aqui utilizamos uma cláusula WHERE (onde). Devemos saber juntamente com o nome do aluno, o nome do curso que ele faz, o que podemos traduzir da seguinte maneira em linguagem SQL: alunos.cod_curso = cursos.id.
Nossa consulta poderá ficar assim:

```
mysql> SELECT * FROM alunos, cursos WHERE cursos.id = alunos.cod_curso;
```

Saída gerada:

```
+----+------------+------+-----------+----+------------+-----------------------------------+------------------+
| id | nome       | obs  | cod_curso | id | nome       | descricao                         | obs              |
+----+------------+------+-----------+----+------------+-----------------------------------+------------------+
|  1 | Juquinha   | NULL |         1 |  1 | MySQL      | Curso Pratico de MySQL            | NULL             |
|  2 | Pedrinho   | NULL |         2 |  2 | Java       | Curso de Linguagem de Programacao | NULL             |
|  3 | Mariazinha | NULL |         3 |  3 | Manutencao | Manutencao de Computadores        | NULL             |
|  4 | Joao       | NULL |         3 |  3 | Manutencao | Manutencao de Computadores        | NULL             |
|  5 | Leia       | NULL |         5 |  5 | J2ME       | Java Micro Edition                | Dois professores |
+----+------------+------+-----------+----+------------+-----------------------------------+------------------+
5 rows in set (0.03 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Agora temos os alunos relacionados apenas com seus cursos. Esta consulta satisfaz nossa necessidade de saber qual aluno faz qual curso. A isto chamamos de INNER JOINS.
Nosso SELECT está retornando todos os campos de todas as tabelas incluídas na consulta (*).

RECURSOS DE VISUALIZAÇÃO:

Podemos limitar os resultados apenas aos campos que desejamos:

```
mysql> SELECT alunos.nome, cursos.descricao, cursos.obs FROM alunos, cursos WHERE cursos.id = alunos.cod_curso;
```

Saída gerada:

```
+------------+-----------------------------------+------------------+
| nome       | descricao                         | obs              |
+------------+-----------------------------------+------------------+
| Juquinha   | Curso Pratico de MySQL            | NULL             |
| Pedrinho   | Curso de Linguagem de Programacao | NULL             |
| Mariazinha | Manutencao de Computadores        | NULL             |
| Joao       | Manutencao de Computadores        | NULL             |
| Leia       | Java Micro Edition                | Dois professores |
+------------+-----------------------------------+------------------+
5 rows in set (0.01 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Também podemos renomear os campos resultantes (recurso chamado alias):

```
mysql> SELECT alunos.nome AS Aluno, cursos.descricao AS Curso, cursos.obs AS Obs FROM alunos, cursos
    WHERE (cursos.id = alunos.cod_curso);
```

Saída gerada:

```
+------------+-----------------------------------+------------------+
| Aluno      | Curso                             | Obs              |
+------------+-----------------------------------+------------------+
| Juquinha   | Curso Pratico de MySQL            | NULL             |
| Pedrinho   | Curso de Linguagem de Programacao | NULL             |
| Mariazinha | Manutencao de Computadores        | NULL             |
| Joao       | Manutencao de Computadores        | NULL             |
| Leia       | Java Micro Edition                | Dois professores |
+------------+-----------------------------------+------------------+
5 rows in set (0.00 sec)
```

Usando a função CONCAT podemos adicionar uma string ao resultado:

```
mysql> SELECT CONCAT("Pessoa: ",alunos.nome) AS Aluno, cursos.descricao AS Curso, cursos.obs AS Obs FROM
    alunos, cursos WHERE (cursos.id = alunos.cod_curso);
```

Saída gerada:

14 – Seleção, cross join, inner join e subseleção:

```
+--------------------+-------------------------------------+-------------------+
| Aluno              | Curso                               | Obs               |
+--------------------+-------------------------------------+-------------------+
| Pessoa: Juquinha   | Curso Pratico de MySQL              | NULL              |
| Pessoa: Pedrinho   | Curso de Linguagem de Programacao   | NULL              |
| Pessoa: Mariazinha | Manutencao de Computadores          | NULL              |
| Pessoa: Joao       | Manutencao de Computadores          | NULL              |
| Pessoa: Leia       | Java Micro Edition                  | Dois professores  |
+--------------------+-------------------------------------+-------------------+
5 rows in set (0.00 sec)
```

SUBSELECTS:

Usamos subselects para organizar nossa consulta. Um subselect (SELECT interno) deve ter apenas um resultado por execução. Este resultado fará parte da seleção (SELECT externo). Veja o exemplo:
A selecao para mostrar os alunos é a seguinte:
SELECT * FROM alunos;
A selecão para mostrar todos os cursos é:
SELECT * FROM cursos;
Para vermos os alunos e os respectivos cursos, podemos fazer a seguinte consulta:
SELECT * FROM alunos, cursos WHERE alunos.cod_curso = cursos.id;
Para sabermos os que fazem Java:
SELECT * FROM alunos, cursos WHERE (alunos.cod_curso = cursos.id) AND (cursos.nome = 'Java');
Podemos colocar a seleção do curso dentro de um subselect, desta forma:

14 – Seleção, cross join, inner join e subseleção:

mysql> SELECT * FROM alunos WHERE cod_curso = (SELECT id FROM cursos WHERE nome = 'Java');

Saída gerada:

```
+----+----------+------+-----------+
| id | nome     | obs  | cod_curso |
+----+----------+------+-----------+
|  2 | Pedrinho | NULL |         2 |
+----+----------+------+-----------+
1 row in set (0.01 sec)
```

Uma consulta que retorne mais de um aluno:

mysql> SELECT * FROM alunos WHERE cod_curso = (SELECT id FROM cursos WHERE nome = 'Manutencao');

Saída gerada:

```
+----+------------+------+-----------+
| id | nome       | obs  | cod_curso |
+----+------------+------+-----------+
|  3 | Mariazinha | NULL |         3 |
|  4 | Joao       | NULL |         3 |
+----+------------+------+-----------+
2 rows in set (0.00 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Um subselect que retorna erro:

mysql> SELECT * FROM alunos WHERE cod_curso = (SELECT id FROM cursos);

Saída gerada:

```
ERROR 1242 (21000): Subquery returns more than 1 row
```

Vejamos o motivo desse erro:

Podemos efetuar uma seleção externa que retorna mais de um resultado. Mas não a seleção interna. A consulta relativa ao curso de Manutenção é válida, pois só existe um curso de manutenção. A última consulta não foi executada, pois existem cinco cursos cadastrados. Vamos ver isso, mostrando os dos últimos subselects:

14 – Seleção, cross join, inner join e subseleção:

mysql> SELECT id FROM cursos WHERE nome = 'Manutencao';

Saída gerada:

```
+----+
| id |
+----+
|  3 |
+----+
1 row in set (0.00 sec)
```

mysql> SELECT id FROM cursos;

Saída gerada:

```
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
+----+
5 rows in set (0.00 sec)
```

Este último, não pode ser aceito, pois só podemos passar para o SELECT pai um resultado. Já o SELECT pai, pode apresentar vários resultados, baseado no retorno do SELECT interno. Exemplo: Curso de Manutenção.

14 – Seleção, cross join, inner join e subseleção:

DISTINCT:

Para quando desejamos forçar a eliminação de duplicidade em um resultado (uma relação). Quando temos mais de um resultado igual, podemos omitir os repetidos, se isto não for atrapalhar. Se o número de repetições não for importante, podemos usar o comando DISTINCT. Veja o seguinte comando que mostra o curso de Manutenção, somente se o curso tiver alunos:

```
mysql> SELECT cursos.nome, cursos.descricao FROM alunos, cursos WHERE cursos.id = alunos.cod_curso AND
    cursos.nome = 'Manutencao';
```

Saída gerada:

```
+------------+----------------------------+
| nome       | descricao                  |
+------------+----------------------------+
| Manutencao | Manutencao de Computadores |
| Manutencao | Manutencao de Computadores |
+------------+----------------------------+
2 rows in set (0.01 sec)
```

Agora com o curso de PHP:

```
mysql> SELECT cursos.nome, cursos.descricao FROM alunos, cursos WHERE cursos.id = alunos.cod_curso AND
    cursos.nome = 'PHP';
```

Saída gerada:

```
Empty set (0.00 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Como o curso de PHP não apresenta alunos, não obemos nenhuma relação.

Existem duas tuplas mostradas no curso de Manutenção, pois duas pessoas fazem esse curso (adicionar alunos.nome para comprovar), João e Mariazinha. Se não houvessem alunos no curso, nenhum resultado seria mostrado (como no de PHP). Mas queremos ver apenas uma vez o nome do curso, e não quantos alunos estão matriculados nesse curso. Utilizando o comando DISTINCT temos apenas um resultado por tuplas iguais geradas:

```
mysql> SELECT DISTINCT cursos.nome, cursos.descricao FROM alunos, cursos WHERE cursos.id = alunos.cod_curso AND cursos.nome = 'Manutencao';
```

Saída gerada:

```
+------------+---------------------------+
| nome       | descricao                 |
+------------+---------------------------+
| Manutencao | Manutencao de Computadores |
+------------+---------------------------+
1 row in set (0.00 sec)
```

Também podemos ver todos os cursos que apresentam alunos:

```
mysql> SELECT DISTINCT cursos.nome, cursos.descricao FROM alunos, cursos WHERE cursos.id = alunos.cod_curso ORDER BY nome;
```

14 – Seleção, cross join, inner join e subseleção:

Saída gerada:

```
+------------+-------------------------------------+
| nome       | descricao                           |
+------------+-------------------------------------+
| J2ME       | Java Micro Edition                  |
| Java       | Curso de Linguagem de Programacao   |
| Manutencao | Manutencao de Computadores          |
| MySQL      | Curso Pratico de MySQL              |
+------------+-------------------------------------+
4 rows in set (0.02 sec)
```

OTIMIZAÇÕES DE CONSULTAS:

Vamos analisar a consulta anterior: Ela nos trás o nome de todos alunos, juntamente com os respectivos cursos. Agora desejamos saber apenas o curso que a Leia faz. A consulta ficaria assim:

mysql> SELECT alunos.nome AS Aluno, cursos.descricao AS Curso, cursos.obs AS Obs FROM alunos, cursos WHERE (cursos.id = alunos.cod_curso) AND (alunos.nome = 'Leia');

Saída gerada:

```
+--------+--------------------+------------------+
| Aluno  | Curso              | Obs              |
+--------+--------------------+------------------+
| Leia   | Java Micro Edition | Dois professores |
+--------+--------------------+------------------+
1 row in set (0.01 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Temos duas cláusulas aqui: O curso que o aluno faz (alunos.cod_curso = cursos.id) e o nome igual a Leia (alunos.nome = 'Leia'). Através do AND garantimos que apenas teremos resultados de alunos relacionados a seus cursos e (AND) alunos com nome igual a Leia.

Observando as cláusulas apresentadas acima, podemos tirar as seguintes conclusões:

* Inicialmente faremos a multiplicação de todas as linhas de alunos por todas as linhas de cursos (5 X 5). São 25 verificações e nossa condição diz que apenas quando alunos.cod_curso for igual a cursos.id o resultado será válido, temos então, 5 valores retornados.
* Após isto, teremos uma nova cláusula. Estes 5 valores retornados, serão novamente verificados. Agora somente quando alunos.nome for igual a Leia, o resultado é aceito. Temos então mais cinco verificações e apenas um resultado válido.
* Então nossa consulta efetuou 30 verificações ao todo.

Se invertermos as cláusulas da consulta, o resultado deverá ser o mesmo:

```
mysql> SELECT alunos.nome AS Aluno, cursos.descricao AS Curso, cursos.obs AS Obs FROM alunos, cursos
    WHERE (alunos.nome = 'Leia') AND (cursos.id = alunos.cod_curso);
```

Saída gerada:

```
+--------+--------------------+------------------+
| Aluno  | Curso              | Obs              |
+--------+--------------------+------------------+
| Leia   | Java Micro Edition | Dois professores |
+--------+--------------------+------------------+
1 row in set (0.00 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Vamos observar novamente o que acontece:

* A primeira cláusula trabalha apenas com a tabela alunos. Serão 5 verificações. Apenas uma pessoa será retornada, Leia (alunos.nome = 'Leia'). Temos 5 verificações e 1 resultado.
* Agora fazermos o INNER JOIN. O resultado da primeira consulta será multiplicado por todos os cursos e apenas quando o alunos.cod_curso for igual ao cursos.id o resultado será válido. Temos mais 5 verificações. Apenas um resultado será aceito. Quando o cursos.id for igual a 5 (id do curso Java ME).
* No total, fizemos 10 verificações.

Concluímos então, que devemos deixar os INNER JOINS para o final da consulta. Assim o servidor pode fazer menos verificações, ajudando na velocidade dos resultados.

ALGUNS EXEMPLOS DE SELECTS DE PROPÓSITO GERAL:

Todas as tabelas e suas chaves primárias no banco (base: information_schema):

```
mysql> SELECT CONCAT(t.table_name,".",t.table_schema) as tbl, c.column_name,c.constraint_name FROM TABLES AS t LEFT JOIN KEY_COLUMN_USAGE AS c ON (t.TABLE_NAME=c.TABLE_NAME AND c.CONSTRAINT_SCHEMA=t.TABLE_SCHEMA AND constraint_name='PRIMARY') WHERE t.table_schema!="information_schema" ORDER BY constraint_name;
```

14 – Seleção, cross join, inner join e subseleção:

Saída gerada:

```
+---------------------------------+---------------------+-----------------+
| tbl                             | column_name         | constraint_name |
+---------------------------------+---------------------+-----------------+
| lyrics.amarok                   | NULL                | NULL            |
| playlists.amarok                | NULL                | NULL            |
| podcastchannels.amarok          | NULL                | NULL            |
| related_artists.amarok          | NULL                | NULL            |
| aluno_curso.teste1              | NULL                | NULL            |
| ...                             | ...                 | ...             |
| ...                             | ...                 | ...             |
| ...                             | ...                 | ...             |
| time_zone_leap_second.mysql     | Transition_time     | PRIMARY         |
| func.mysql                      | name                | PRIMARY         |
+---------------------------------+---------------------+-----------------+
63 rows in set (0.82 sec)
```

Mostrar usuários, hosts e suas senha do banco de dados (base: mysql):

mysql> SELECT Host, User, Password FROM user;

Saída gerada:

```
+-----------+------------------+-------------------------------------------+
| Host      | User             | Password                                  |
+-----------+------------------+-------------------------------------------+
| localhost | root             | *983A12F05F117E057023CDD4E24A51E433ACA334 |
| house     | root             | *983A12F05F117E057023CDD4E24A51E433ACA334 |
| 127.0.0.1 | root             | *983A12F05F117E057023CDD4E24A51E433ACA334 |
| localhost | debian-sys-maint | *983A12F05F117E057023CDD4E24A51E433ACA334 |
+-----------+------------------+-------------------------------------------+
4 rows in set (0.00 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Ver lista de palavras que apresentam ajuda (HELP) (base: mysql):

mysql> SELECT * FROM help_keyword ORDER BY name;

Saída gerada:

```
+-----------------+---------------------------+
| help_keyword_id | name                      |
+-----------------+---------------------------+
|             382 | <>                        |
|             363 | ACTION                    |
|             139 | ADD                       |
|             332 | AES_ENCRYPT               |
|             184 | AFTER                     |
|             ... | ...                       |
|             ... | ...                       |
|             ... | ...                       |
|              25 | YEAR_MONTH                |
|             205 | ZEROFILL                  |
+-----------------+---------------------------+
395 rows in set (0.00 sec)
```

VISÕES:

Podemos definir visões como seleções pré-definidas. Não fazem parte do modelo lógico do banco de dados. Elas podem ser muito úteis para simplificar o trabalho de repetir consultas (ou até mesmo errar algo). Vamos fazer uma visão simples, mostrando apenas o nome do aluno e o seu curso.

14 – Seleção, cross join, inner join e subseleção:

mysql> CREATE VIEW aluno_curso AS SELECT alunos.nome AS Aluno, cursos.descricao AS Curso FROM alunos, cursos WHERE cursos.id = alunos.cod_curso;

Saída gerada:

```
Query OK, 0 rows affected (0.13 sec)
```

Agora podemos simplificar nossas consultas acessando apenas a visão (VIEW) que realiza a seleção correta:

mysql> SELECT * FROM aluno_curso;

Saída gerada:

```
+------------+-----------------------------------+
| Aluno      | Curso                             |
+------------+-----------------------------------+
| Juquinha   | Curso Pratico de MySQL            |
| Pedrinho   | Curso de Linguagem de Programacao |
| Mariazinha | Manutencao de Computadores        |
| Joao       | Manutencao de Computadores        |
| Leia       | Java Micro Edition                |
+------------+-----------------------------------+
5 rows in set (0.09 sec)
```

14 – Seleção, cross join, inner join e subseleção:

RENOMEAÇÃO DE TABELAS:

Podemos poupar tempo durante uma consulta através de renomeações de tabelas. Na clausula WHERE informamos a tabela e após atribuimos um apelido, agora durante as seleções na clausula SELECT podemos utilizar o apelido seguido da coluna desejada. Em consultas pequenas isso não trás benefícios, mas para consultas que envolvam várias tabelas é interessante utilizar este recurso.

mysql> SELECT a.nome, a.obs FROM alunos a;

Saída gerada:

```
+------------+------+
| nome       | obs  |
+------------+------+
| Juquinha   | NULL |
| Pedrinho   | NULL |
| Mariazinha | NULL |
| Joao       | NULL |
| Leia       | NULL |
| Paulinha   | NULL |
+------------+------+
6 rows in set (0.00 sec)
```

14 – Seleção, cross join, inner join e subseleção:

LIMITES:

Para quando temos muitos resultados e nem todos nos interessam, podemos limitar os resultados apresentados. Por exemplo, se desejamos verificar os nomes da lista de alunos em order alfabética, podemos executar a consulta:

mysql> SELECT a.nome, a.obs FROM alunos a ORDER BY nome;

Saída gerada:

```
+------------+------+
| nome       | obs  |
+------------+------+
| Joao       | NULL |
| Juquinha   | NULL |
| Leia       | NULL |
| Mariazinha | NULL |
| Paulinha   | NULL |
| Pedrinho   | NULL |
+------------+------+
6 rows in set (0.00 sec)
```

14 – Seleção, cross join, inner join e subseleção:

Agora para vermos apenas os primeiros alunos:

mysql> SELECT a.nome, a.obs FROM alunos a ORDER BY nome LIMIT 3;

Saída gerada:

```
+----------+------+
| nome     | obs  |
+----------+------+
| Joao     | NULL |
| Juquinha | NULL |
| Leia     | NULL |
+----------+------+
3 rows in set (0.00 sec)
```

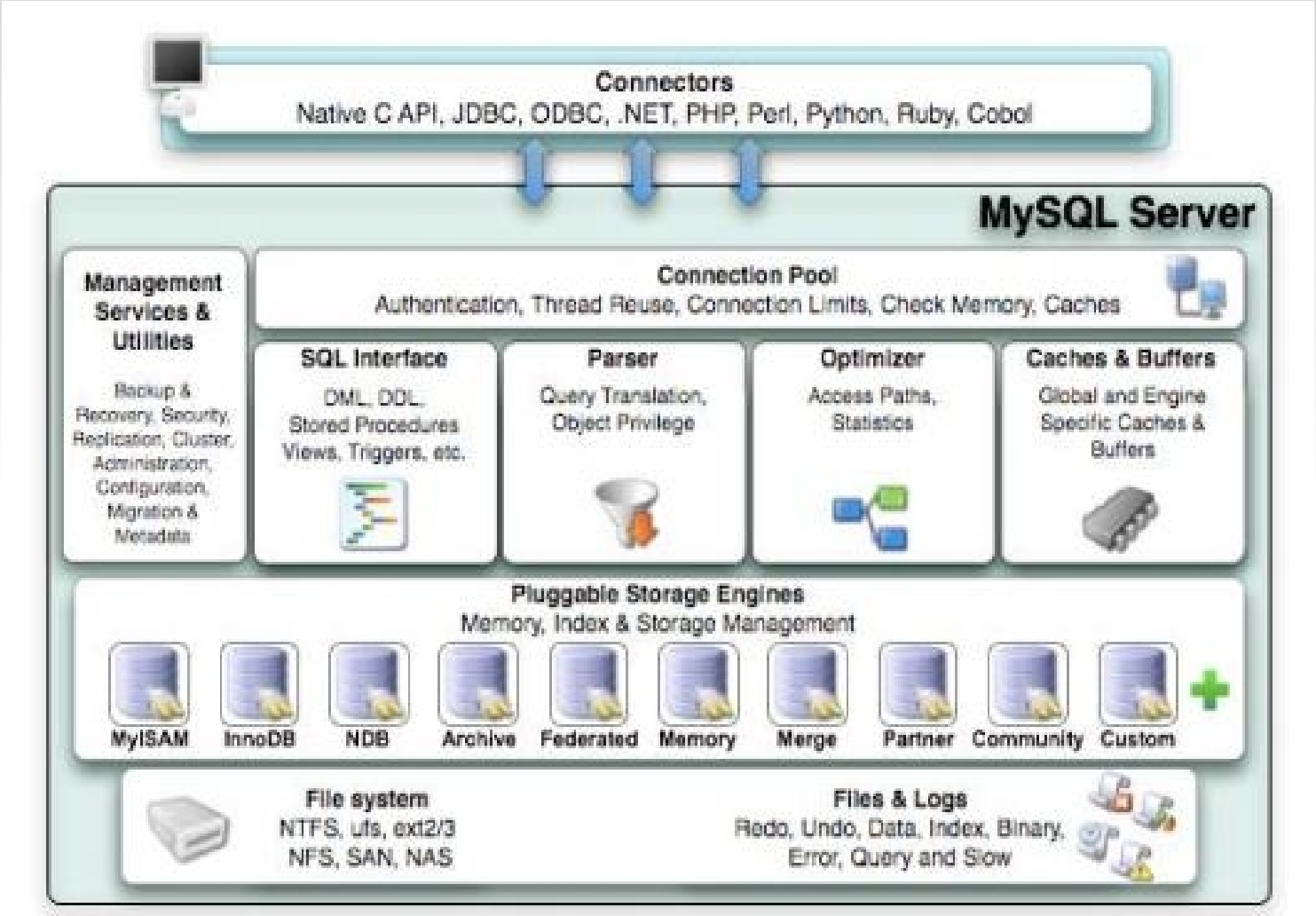
15 – Tipos de tabelas:

mysql> SHOW ENGINES;

Saída gerada:

```
+------------+----------+----------------------------------------------------------------+
| Engine     | Support  | Comment                                                        |
+------------+----------+----------------------------------------------------------------+
| MyISAM     | DEFAULT  | Default engine as of MySQL 3.23 with great performance         |
| MEMORY     | YES      | Hash based, stored in memory, useful for temporary tables      |
| InnoDB     | YES      | Supports transactions, row-level locking, and foreign keys     |
| BerkeleyDB | NO       | Supports transactions and page-level locking                   |
| BLACKHOLE  | YES      | /dev/null storage engine (anything you write to it disappears) |
| EXAMPLE    | NO       | Example storage engine                                         |
| ARCHIVE    | YES      | Archive storage engine                                         |
| CSV        | YES      | CSV storage engine                                             |
| ndbcluster | DISABLED | Clustered, fault-tolerant, memory-based tables                 |
| FEDERATED  | YES      | Federated MySQL storage engine                                 |
| MRG_MYISAM | YES      | Collection of identical MyISAM tables                          |
| ISAM       | NO       | Obsolete storage engine                                        |
+------------+----------+----------------------------------------------------------------+
12 rows in set (0.01 sec)
```

15 – Tipos de tabelas:



15 – Tipos de tabelas:

Os tipos de tabelas são os modelos de engenharia de armazenamento de dados do banco MySQL. Seu servidor é muito robusto e apresenta diversos tipos de tabelas. Cada tabela apresenta sua maneira de arquivar os dados (existe até mesmo um tipo que não faz armazenamento de dados). O Servidor do MySQL é muito versátil. Graças ao seus sistemas de tipos de tabelas, pode desempenhar diversas funções em vários tipos de aplicações. Vejamos agora alguns tipos de tabelas:

MEMORY (Memory Storage Engine):

Utilizado para consultas muito rápidas. O tipo Memory na verdade é apenas um esqueleto de tipo de dados. Grava apenas dados na memória, não sendo gravados no servidor. São tabelas temporárias usadas para aumentar a performace e a velocidade das consultas (tabela do tipo Hash Table).

MYISAM:

Alto desempenho para aplicações que realizam tarefas de saída de dados do servidor (consultas). Para leitura e escrita outros tipos de tabelas se mostram mais eficazes. Durante uma inserção na tabela o MyISAM faz um bloqueio total a tabela (conhecido como lock simples). Tipo de tabela defaul do MySQL.

ARCHIVE (Archive Storage Engine):

Tipo de tabela utilizada para armazenar grandes quantidades de dados. Aceita apenas os comandos INSERT e SELECT. Faz a compressão dos dados ao inserir. Para remover, deve-se apagar os arquivos gerados por este tipo de tabela. Não apresenta índices.

15 – Tipos de tabelas:

MERGE (Merge Storage Engine):

Faz a junção de várias tabelas do tipo MyISAM, possibilitando o uso delas como se fosse uma única tabela.

INNODB (InnoDB Storage Engine):

Único tipo de tabelas do MySQL que apresenta as características ACID:
Atomicidade (atômico, indivisível): Apresenta controle das transações (COMMIT e ROLLBACK). Garante que a transação terá todas as operações realizadas ou nenhuma será validada.
Consistência: Chaves primárias e estrangeiras são respeitadas neste tipo de tabela.
Isolamento: Transações executando ao mesmo tempo (paralelas) não são vistas umas pelas outras, apenas quando seus COMMIT's são executados.
Durabilidade: Após um COMMIT os dados estão gravados de forma concreta no banco, mesmo que falhas ocorram posteriormente.
Aceita um grande número de acessos e gravações de dados simultaneamente com um ótimo desempenho.

CLUSTER (Cluster Storage Engine):

Este tipo multiplica as tabelas (replica) em vários servidores (clusterização). Este tipo é utilizado em aplicações críticas (ou seja, não pode haver travamento do banco de dados). É o tipo com melhor garantia de armazenamento de dados do MySQL. Caso um servidor páre, os outros continuam trabalhando normalmente. Sistema muito rápido. Seu sistema de armazenamento pode ser Hash Table ou Árvores. Conhecido como Cinco Noves, isto é, 99,999% de disponibilidade.

16 – Nossa aplicação:

Trabalharemos com a modelação de um programa real para explicarmos melhor os conceitos passados até aqui e ensinarmos algumas novas lições. Vamos analizar o banco de dados de um programa de biblioteca de músicas. O AMAROK é um player de música open source que faz o gerenciamento, catalogando em bandas e álbuns as músicas selecionadas. Este programa pode usar três banco de dados (MySQL, PostgreSQL e SQLITE). Por padrão o AMAROK utiliza o SQLITE, mas se instalarmos o MySQL podemos trocar o banco de dados. Aqui está uma amostra do banco original:

mysql> use amarok

Saída gerada:

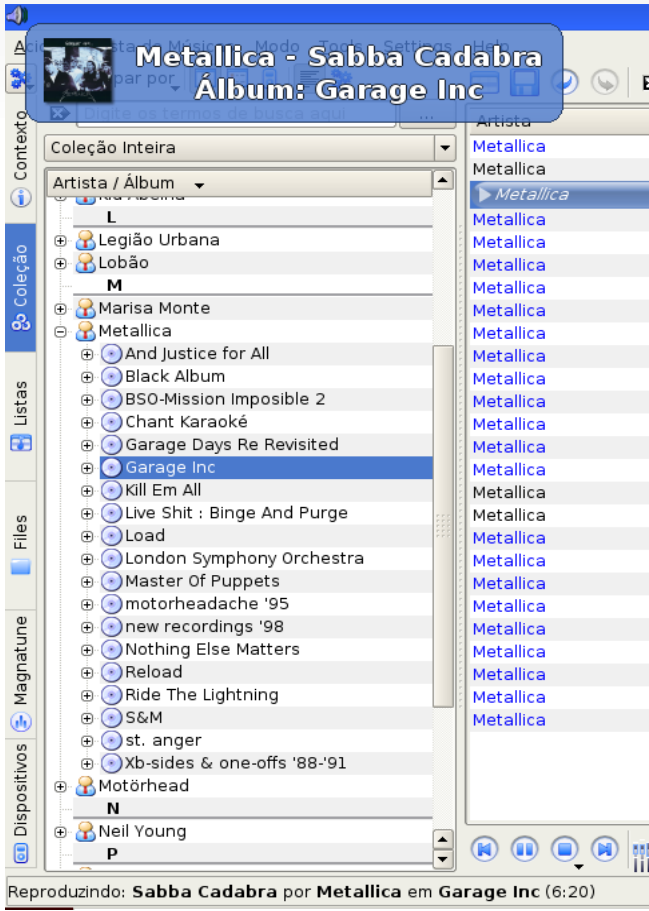
```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

16 – Nossa aplicação:

Janela de contexto com informações das músicas e janela com a lista de reprodução

Artistas e álbuns



16 – Nossa aplicação:

mysql> show tables;

Saída gerada:

```
+------------------+
| Tables_in_amarok |
+------------------+
| admin            |
| album            |
| amazon           |
| artist           |
| composer         |
| devices          |
| directories      |
| embed            |
| genre            |
| images           |
| labels           |
| lyrics           |
| playlists        |
| podcastchannels  |
| podcastepisodes  |
| podcastfolders   |
| related_artists  |
| statistics       |
| tags             |
| tags_labels      |
| uniqueid         |
| year             |
+------------------+
22 rows in set (0.00 sec)
```

16 – Nossa aplicação:

Vamos analizar as tabelas principais (com o comando SHOW FIELDS ou DESCRIBE):

mysql> SHOW FIELDS FROM album;

Saída gerada:

```
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(255) | YES  | MUL | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.01 sec)
```

mysql> SHOW FIELDS FROM artist;

Saída gerada:

```
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(255) | YES  | MUL | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

16 – Nossa aplicação:

O sistema gira em torno das músicas cadastradas. Veja a tabela de músicas:

mysql> SHOW INDEXES FROM tags;

Saída gerada:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment
tags	0	url_tag	1	url	A	NULL	NULL	NULL	YES	BTREE	
tags	0	url_tag	2	deviceid	A	NULL	NULL	NULL	YES	BTREE	
tags	1	album_tag	1	album	A	274	NULL	NULL	YES	BTREE	
tags	1	artist_tag	1	artist	A	21	NULL	NULL	YES	BTREE	
tags	1	composer_tag	1	composer	A	83	NULL	NULL	YES	BTREE	
tags	1	genre_tag	1	genre	A	30	NULL	NULL	YES	BTREE	
tags	1	year_tag	1	year	A	39	NULL	NULL	YES	BTREE	
tags	1	sampler_tag	1	sampler	A	1	NULL	NULL	YES	BTREE	
tags	1	tags_artist_index	1	artist	A	21	NULL	NULL	YES	BTREE	
tags	1	tags_album_index	1	album	A	274	NULL	NULL	YES	BTREE	
tags	1	tags_deviceid_index	1	deviceid	A	1	NULL	NULL	YES	BTREE	
tags	1	tags_url_index	1	url	A	3019	NULL	NULL	YES	BTREE	

12 rows in set (0.00 sec)

16 – Nossa aplicação:

Vamos montar nosso sistema:

mysql> CREATE DATABASE biblioteca_musical;

Saída gerada:

```
Query OK, 1 row affected (0.00 sec)
```

mysql> USE biblioteca_musical

Saída gerada:

```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

16 – Nossa aplicação:

```
mysql> CREATE TABLE album (
    -> id int auto_increment,
    -> nome varchar(255),
    -> CONSTRAINT chave_pri_album PRIMARY KEY (id)
    -> ) ENGINE = INNODB;
```

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

mysql> SHOW FIELDS FROM album;

Saída gerada:

```
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| nome  | varchar(255) | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

mysql> SHOW INDEXES FROM album;

Saída gerada:

```
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
| album |          0 | PRIMARY  |            1 | id          | A         |           0 |     NULL | NULL   |      | BTREE      |         |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
1 row in set (0.00 sec)
```

16 – Nossa aplicação:

Apenas a tabela álbum criada:

16 – Nossa aplicação:

```
mysql> CREATE TABLE compositor (
    -> id int auto_increment,
    -> nome varchar(255),
    -> CONSTRAINT chave_pri_comp PRIMARY KEY (id)
    -> ) ENGINE = INNODB;
```

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE TABLE artista (
    -> id int auto_increment,
    -> nome varchar(255),
    -> CONSTRAINT chave_pri_artist PRIMARY KEY (id)
    -> ) ENGINE = INNODB;
```

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE TABLE genero (
    -> id int auto_increment,
    -> nome varchar(255),
    -> CONSTRAINT chave_pri_gen PRIMARY KEY (id)
    -> ) ENGINE = INNODB;
```

Saída gerada:

```
Query OK, 0 rows affected (0.07 sec)
```

16 – Nossa aplicação:

As quatro primeiras tabelas:

ÁLBUM
id
nome

GENERO
id
nome

ARTISTA
id
nome

COMPOSITOR
id
nome

16 – Nossa aplicação:

A tabela de ano de lançamento no AMAROK utiliza uma chave primária e a descrição do ano:

mysql> SHOW FIELDS FROM year;

Saída gerada:

```
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(255) | YES  | MUL | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

Nosso exemplo utilizará o ano como chave primária:

```
mysql> CREATE TABLE ano (
    -> id int  auto_increment,
    -> CONSTRAINT chave_pri_artist PRIMARY KEY (id)
    -> ) ENGINE = INNODB;
```

Saída gedada:

```
Query OK, 0 rows affected (0.01 sec)
```

16 – Nossa aplicação:

mysql> SHOW FIELDS FROM ano;

Saída gerada:

```
+-------+---------+------+-----+---------+----------------+
| Field | Type    | Null | Key | Default | Extra          |
+-------+---------+------+-----+---------+----------------+
| id    | int(11) | NO   | PRI | NULL    | auto_increment |
+-------+---------+------+-----+---------+----------------+
1 row in set (0.01 sec)
```

mysql> SHOW INDEXES FROM ano;

Saída gerada:

```
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
| ano   |          0 | PRIMARY  |            1 | id          | A         |           0 |     NULL | NULL   |      | BTREE      |         |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+
1 row in set (0.00 sec)
```

16 – Nossa aplicação:

Tabela ano criada:

ANO
id

ÁLBUM
id
nome

GENERO
id
nome

ARTISTA
id
nome

COMPOSITOR
id
nome

16 – Nossa aplicação:

O sistema de banco de dados do AMAROK armazena o ano por música, mas nosso sistema armazenará o ano baseado no álbum. Façamos as alterações na nossa tabela album (ATER TABLE):

mysql> ALTER TABLE album ADD COLUMN cod_ano int AFTER nome;

Saída gerada:

```
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Agora já contamos com a coluna ano na tabela album, mas falta a chave estrangeira:

mysql> ALTER TABLE album ADD CONSTRAINT chave_ext_ano_album FOREIGN KEY (cod_ano) REFERENCES ano(id);

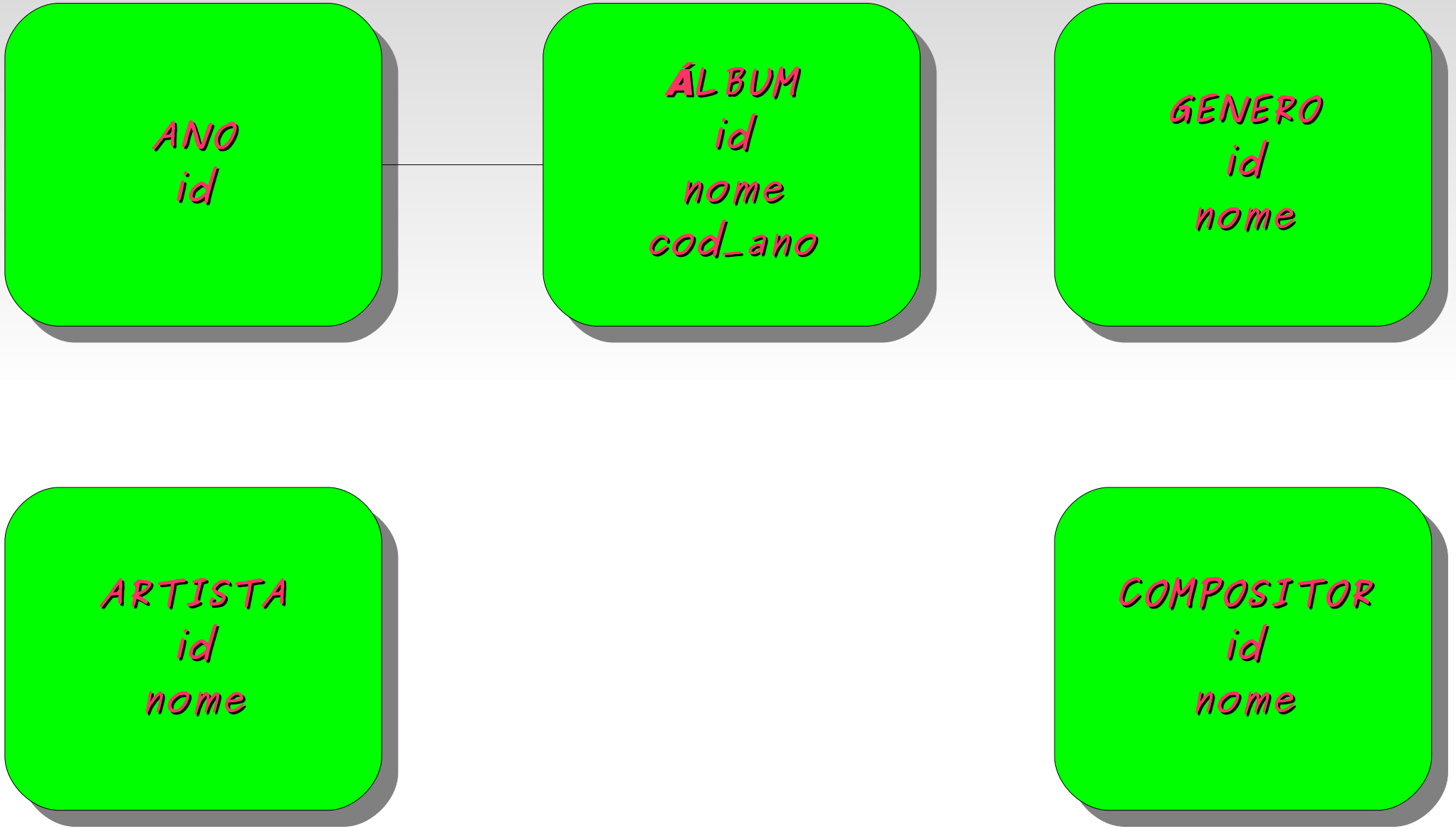
Saída gerada:

```
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Executando o comando SHOW podemos ver a coluna nova e também a chave estrangeira.

16 – Nossa aplicação:

Primeira chave estrangeira criada:



16 – Nossa aplicação:

Nossa tabela album terá um (e apenas um) gênero. Então cada tupla na tabela album terá apenas um gênero. Fará ligação ao código do gênero.

mysql> ALTER TABLE album ADD COLUMN cod_genero int AFTER cod_ano;

Saída gerada:

```
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

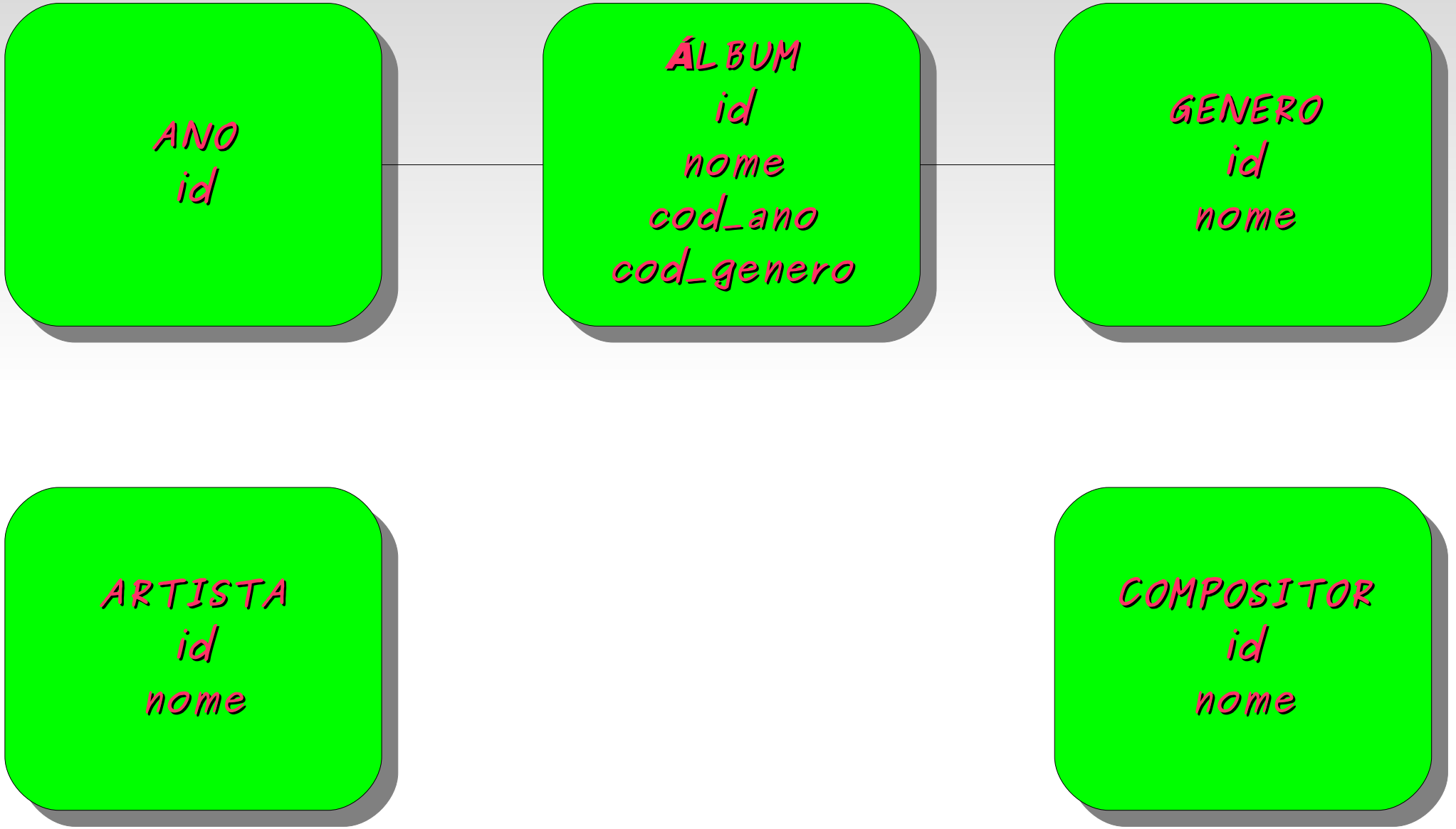
mysql> ALTER TABLE album ADD CONSTRAINT chave_ext_gen_album FOREIGN KEY (cod_genero) REFERENCES genero(id);

Saída gerada:

```
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

16 – Nossa aplicação:

Relacionamento entre o álbum e o gênero.



16 – Nossa aplicação:

Tabela música:
A tabela música também terá os mesmos campos comuns das demais (id e nome). A tabela terá algumas chaves estrangeiras:
* Cada música terá apenas um artista (em caso de mais, haverá uma tupla na tabela artista com esses artistas). Sempre haverá apenas uma ligação entre a música e o artista. Então terá uma chave estrangeira para o artista.
* Cada música terá apenas um compositor. Haverá uma chave estrangeira para a tabela compositor.
* Cada música fará parte de um único álbum. Cada música cadastrada deverá ser associada a um (e apenas um) álbum. Também haverá uma chave estrangeira para a tabela album.

Nossa tabela apresenta: Uma chave primária e três chaves estrangeiras.

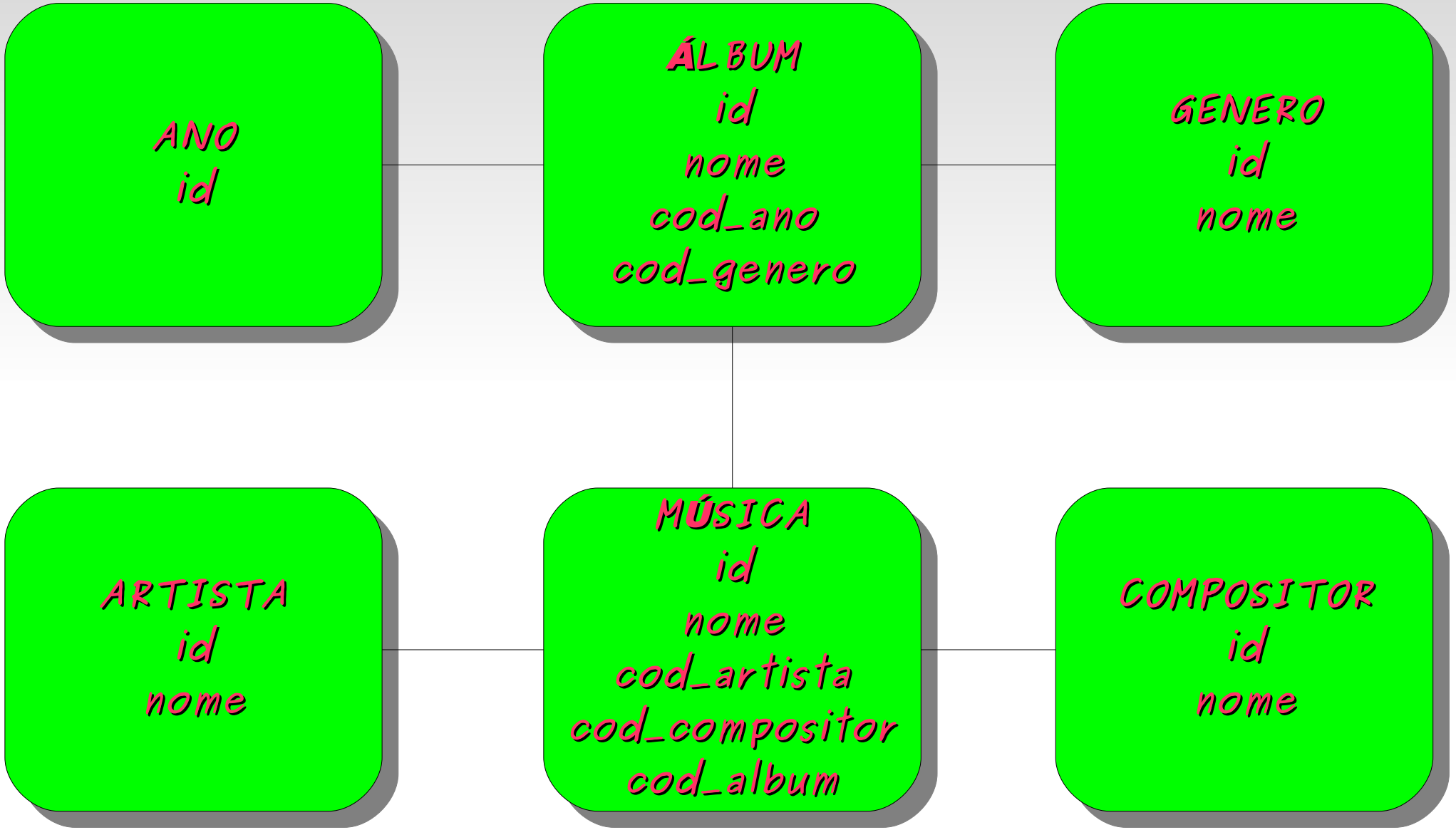
```
mysql> CREATE TABLE musica (
    -> id int auto_increment,
    -> nome varchar(255),
    -> cod_artista int,
    -> cod_compositor int,
    -> cod_album int,
    -> CONSTRAINT chave_pri_mus PRIMARY KEY (id),
    -> CONSTRAINT chave_art_mus FOREIGN KEY (cod_artista) REFERENCES artista(id),
    -> CONSTRAINT chave_com_mus FOREIGN KEY (cod_compositor) REFERENCES compositor(id),
    -> CONSTRAINT chave_album_mus FOREIGN KEY (cod_album) REFERENCES album(id)
    -> ) ENGINE = INNODB;
```

Saída gerada:

```
Query OK, 0 rows affected (0.66 sec)
```

16 – Nossa aplicação:

As tabelas e seus relacionamentos:



16 – Nossa aplicação:

mysql> SHOW FIELDS FROM musica;

Saída gerada:

```
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| id              | int(11)      | NO   | PRI | NULL    | auto_increment |
| nome            | varchar(255) | YES  |     | NULL    |                |
| cod_artista     | int(11)      | YES  | MUL | NULL    |                |
| cod_compositor  | int(11)      | YES  | MUL | NULL    |                |
| cod_album       | int(11)      | YES  | MUL | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
5 rows in set (0.25 sec)
```

mysql> SHOW INDEXES FROM musica;

Saída gerada:

```
+--------+------------+----------------+--------------+---------------+-----------+-------------+----------+--------+------+------------+---------
    +
| Table  | Non_unique | Key_name       | Seq_in_index | Column_name   | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment
    |
+--------+------------+----------------+--------------+---------------+-----------+-------------+----------+--------+------+------------+---------
    +
| musica |          0 | PRIMARY        |            1 | id            | A         |           0 |     NULL | NULL   |      | BTREE      |
    |
| musica |          1 | chave_art_mus  |            1 | cod_artista   | A         |           0 |     NULL | NULL   | YES  | BTREE      |
    |
| musica |          1 | chave_com_mus  |            1 | cod_compositor| A         |           0 |     NULL | NULL   | YES  | BTREE      |
    |
| musica |          1 | chave_album_mus|            1 | cod_album     | A         |           0 |     NULL | NULL   | YES  | BTREE      |
    |
+--------+------------+----------------+--------------+---------------+-----------+-------------+----------+--------+------+------------+---------
    +
4 rows in set (0.10 sec)
```

16 – Nossa aplicação:

Agora nosso sistema já está com o seu esqueleto principal montado. Falta preencher com alguns dados. Se analizarmos o comportamento das tabelas, vamos perceber que não devemos inserir uma música sem antes termos no banco um artista, um compositor e um álbum (na verdade, é possível fazer inserções em musica, pois não criamos nossas chaves estrangeiras como NOT NULL). Não devemos também inserir dados na tabela album, sem antes termos o ano cadastrado e também um gênero para o álbum a ser gravado. Este controle cabe à aplicação que se comunica com o banco. Como nosso objetivo não é estudar o aplicativo AMAROK, vamos fazer algumas inserções manuais:

16 – Nossa aplicação:

mysql> INSERT INTO ano VALUES (1996), (1997), (1998), (1999), (2000);

Saída gerada:

```
Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

mysql> INSERT INTO genero (nome) VALUES ('Heavy Metal'), ('Rock'), ('Pop');

Saída gerada:

```
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

mysql> INSERT INTO artista (nome) VALUES ('Black Eyed Peas'), ('Bon Jovi'), ('Creedence Clearwater Revival'), ('David Coverdale'), ('Demons & Wizards'), ('Dinosaur Jr.'), ('Dream Theater'), ('Fergie'), ('Genesis'), ('Hammerfall'), ('Iggy Pop'), ('Kid Abelha'), ('Legend'), ('Legião Urbana'), ('Lobão'), ('Los Hermanos'), ('Marisa Monte'), ('Metallica'), ('Motörhead'), ('Neil Young'), ('Paula Toller'), ('Paw'), ('Pet Shop Boys'), ('Pitty'), ('Soundgarden'), ('Stratovarius'), ('Temple Of The Dog'), ('Tequila Baby'), ('The Cranberries'), ('Tribalistas'), ('Whitesnake');

Saída gerada:

```
Query OK, 31 rows affected (0.02 sec)
Records: 31  Duplicates: 0  Warnings: 0
```

16 – Nossa aplicação:

Temos no sistema, vários artistas, três gêneros e algumas datas. Vamos popular a tabela de álbums. Vamos inserir alguns álbuns:

mysql> INSERT INTO album (nome) VALUES ('Stratovarius'), ('Elements Pt. I'), ('Elements Pt. II'), ('Intermission'), ('The Chosen Ones'), ('Dreamspace'), ('Singles B-Sides Cd1'), ('Singles B-Sides Cd2'), ('Fright Night'), ('Episode'), ('Infinite'), ('Twilight Time'), ('Visions'), ('Destiny'), ('Fourth Dimension');

Saída gerada:

```
Query OK, 15 rows affected (0.05 sec)
Records: 15  Duplicates: 0  Warnings: 0
```

Agora cadastraremos as músicas do álbum Destiny da banda Stratovarius:

mysql> INSERT INTO musica (nome, cod_artista, cod_album) VALUES ('Cold Winter Nights (Bonus Trac', 26, 14), ('SOS', 26, 14), ('No Turning Back', 26, 14), ('Rebel', 26, 14), ('Playing with Fire', 26, 14), ('Venus in the Morning', 26, 14), ('4000 Rainy Nights', 26, 14), ('Anthem of the World', 26, 14), ('Years Go By', 26, 14), ('Destiny', 26, 14);

Saída gerada:

```
Query OK, 10 rows affected (0.04 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

16 – Nossa aplicação:

Vamos analisar o que faz essa última inserção: O nome da música é passado no primeiro parâmetro inserido, o artista (Stratovarius) tem o código 26 e o álbum (Destiny) tem código 14.

Se efetuarmos uma seleção na tabela album, veremos que nossos registros não apresentam ano e também não apresentam uma ligação para o gênero. Façamos agora uma ligação para o gênero do nosso álbum cadastrado: Vamos saber qual é o código ao certo:

mysql> SELECT * FROM genero;

Saída gerada:

```
+----+-------------+
| id | nome        |
+----+-------------+
|  1 | Heavy Metal |
|  2 | Rock        |
|  3 | Pop         |
+----+-------------+
3 rows in set (0.00 sec)
```

16 – Nossa aplicação:

O gênero do álbum Destiny é o de código 1.

mysql> SELECT * FROM album;

Saída gerada:

```
+----+---------------------+---------+------------+
| id | nome                | cod_ano | cod_genero |
+----+---------------------+---------+------------+
|  1 | Stratovarius        |    NULL |       NULL |
|  2 | Elements Pt. I      |    NULL |       NULL |
|  3 | Elements Pt. II     |    NULL |       NULL |
|  4 | Intermission        |    NULL |       NULL |
|  5 | The Chosen Ones     |    NULL |       NULL |
|  6 | Dreamspace          |    NULL |       NULL |
|  7 | Singles B-Sides Cd1 |    NULL |       NULL |
|  8 | Singles B-Sides Cd2 |    NULL |       NULL |
|  9 | Fright Night        |    NULL |       NULL |
| 10 | Episode             |    NULL |       NULL |
| 11 | Infinite            |    NULL |       NULL |
| 12 | Twilight Time       |    NULL |       NULL |
| 13 | Visions             |    NULL |       NULL |
| 14 | Destiny             |    NULL |       NULL |
| 15 | Fourth Dimension    |    NULL |       NULL |
+----+---------------------+---------+------------+
15 rows in set (0.00 sec)
```

16 – Nossa aplicação:

O álbum está cadastrado com o código número 14. Agora podemos fazer as suas atualizações:

mysql> UPDATE album SET cod_genero = 1 WHERE album.id = 14;

Saída gerada:

```
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

mysql> UPDATE album SET cod_ano = 1998 WHERE album.id = 14;

Saída gerada:

```
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Vamos ver como ficou nossa tupla do álbum Destiny:

mysql> SELECT * FROM album WHERE album.nome = 'Destiny';

Saída gerada:

```
+----+---------+---------+------------+
| id | nome    | cod_ano | cod_genero |
+----+---------+---------+------------+
| 14 | Destiny |    1998 |          1 |
+----+---------+---------+------------+
1 row in set (0.00 sec)
```

17 – Realizando backups:

A principal ferramente de backups utilizada no MySQL é o mysqldump. Aplicativo que acompanha o pacote de programas do MySQL, o mysqldump é basicamente um cliente que grava o conteúdo de tabelas em um arquivo. Normalmente, para acessarmos a ferramenta de backup, devemos utilizar as mesmas opções que usamos para acessar o cliente de SQL do banco de dados (por exemplo, -P, -p, -u, etc).

O mysqldump permite fazer cópias de segurança de todos os bancos de dados do servidor, de alguns bancos, ou apenas de algumas tabelas de determinados bancos.

Por exemplo, vamos fazer o backup do nosso banco (devemos estar fora do cliente mysql), utilizamos o comando mysqldump, seguido das opções normais e após o nome do banco de dados e uma indicação para o arquivo de backup:

```
julio@house:~/MySQL$ mysqldump -u root -p -h localhost biblioteca_musical > BIBLIOTECA.sql
```

Para fazermos o backup de uma tabela, adicionamos o nome da tabela após o banco:

```
julio@house:~/MySQL$ mysqldump -u root -p -h localhost biblioteca_musical artista > ARTISTA.sql
```

Para mais de uma tabela, indicamos os seus nomes após o banco:

```
julio@house:~/MySQL$ mysqldump -u root -p -h localhost biblioteca_musical artista genero > TABELAS.sql
```

Com a seta, especificamos um arquivo para a saída de nosso backup, chamado BIBLIOTECA.sql. Para efetuar a cópia de todas a base de dados do AMAROK, utiliza-se o comando (reduzido):

```
julio@house:~/MySQL$ mysqldump -u root -p amarok > AMAROK.SQL
```

17 – Realizando backups:

Por padrão após efetuarmos um backup, limpamos os logs do sistema. Dessa forma teremos um relatório de logs limpo. Ficando fácil monitorar todas as atividades ocorridas após o backup até uma possivel falha do sistema.

mysql> FLUSH LOGS;

Embora no nosso caso não tenhamos utilizado muitas inserções e remoções no banco, é interessante em tabelas do tipo INNODB, desabilitarmos as chaves (exemplo: ALTER TABLE tags DISABLE KEYS;), pois esse sistema é o único que avalia as chaves estrangeiras. O que o mysqldump faz é uma cópia do sistema com está no momento e não uma recaptulação de todos os comandos dados no banco.

Para retornarmos uma base de dados a um backup, podemos utilizar o cliente de SQL (mysql), apenas indicando que arquivo queremos carregar para a base especificada:

mysql> CREATE DATABASE amarok2;

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
```

julio@house:~/MySQL$ mysql -u root -p -D amarok2 < AMAROK.SQL

17 – Realizando backups:

Copiamos todas as tabelas e todos as suas tuplas para o banco de dados amarok2. Nossa tabela receptora estava vazia, mas esta poderia estar com as tabelas e até mesmos com dados cadastrados. O mysqldump monta as tabelas e copia os dados (dessa forma podemos copiar o conteúdo de segurança para uma aplicação vazia). No momento de importarmos os dados (através do cliente mysql) o mysql irá apagando e recriando as tabelas que estiverem no backup. Todos os dados de uma tabela reposta serão perdidos. Serão gravados os dados do backup. Para gravar, o cliente trava a tabela e insere os dados, após ela é liberada. Este processo de trancamento de tabelas garante que elas ficarão exatamente como estavam no momento do backup. Os únicos dados do banco receptor que não serão perdidos, serão apenas os de tabelas que não constam no backup. Essas tabelas (e seus dados) são mantidas. Ou seja, o processo de recuperação de um backup não deleta todas as tabelas, apenas as que estiverem no backup.

18 – Utilizando vários bancos de dados:

Agora podemos andar e buscar informações nos bancos de dados. Vamos entrar no nosso banco amarok2 adicionando o parâmetro "-s" para visualizar os resultados de maneira simples. Embora estejamos na base de dados amarok2, podemos ver o conteúdo de outras bases. Por exemplo:

mysql> SELECT nome FROM biblioteca_musical.musica;

Saída gerada:

```
nome
Cold Winter Nights (Bonus Trac
SOS
No Turning Back
Rebel
Playing with Fire
Venus in the Morning
4000 Rainy Nights
Anthem of the World
Years Go By
Destiny
```

18 – Utilizando vários bancos de dados:

Vamos buscar os generos da base de dados amarok2 e copia-los para nossa base biblioteca_musical. Podemos fazer uma consulta normal. Exemplo:

mysql> SELECT * FROM genre;

Saída gerada:

```
id      name
1       Pop
2       Rock/Pop
3       MPB
4
5       Latin
6       Blues
7       pop
8       Rap
9       rock
10      Pop/Hair Metal
11      Other
12      Rock
13      Hard Rock
14      Progressive Rock
15      Miscellaneous
16      Brazil
17      Alt. Rock
18      genre
19      misc
20      Heavy Metal
21      Metal
22      Rock & Roll
23      Métal
24      Karaoké
25      Sound Track
26      Folk Rock
27      Punk Rock
28      Unknown
29      Pop/Rock
31      Power Metal
```

18 – Utilizando vários bancos de dados:

Obtemos um resultado vazio (id = 4), vamos omitir esse valor:

mysql> SELECT * FROM genre WHERE name != '';

Saída gerada:

```
id      name
1       Pop
2       Rock/Pop
3       MPB
5       Latin
6       Blues
7       pop
8       Rap
9       rock
10      Pop/Hair Metal
11      Other
12      Rock
13      Hard Rock
14      Progressive Rock
15      Miscellaneous
16      Brazil
17      Alt. Rock
18      genre
19      misc
20      Heavy Metal
21      Metal
22      Rock & Roll
23      Métal
24      Karaoké
25      Sound Track
26      Folk Rock
27      Punk Rock
28      Unknown
29      Pop/Rock
31      Power Metal
```

18 – Utilizando vários bancos de dados:

Vamos utilizar apenas o nome do gênero para copiá-lo. Vamos concatenar o nome com as aspas usadas para inserir strings e os parênteses:

mysql> SELECT CONCAT("('", name, "'),") FROM genre WHERE name != '';

Saída gerada:

```
CONCAT("('",  name, "')," )
('Alt. Rock'),
('Blues'),
('Brazil'),
('Folk Rock'),
('genre'),
('Hard Rock'),
('Heavy Metal'),
('Karaoké'),
('Latin'),
('Métal'),
('Metal'),
('misc'),
('Miscellaneous'),
('MPB'),
('Other'),
('Pop'),
('pop'),
('Pop/Hair Metal'),
('Pop/Rock'),
('Power Metal'),
('Progressive Rock'),
('Punk Rock'),
('Rap'),
('rock'),
('Rock'),
('Rock & Roll'),
('Rock/Pop'),
('Sound Track'),
('Unknown'),
```

18 – Utilizando vários bancos de dados:

Basta trocar o último caracter (a vírgula) por um ponto-e-vírgula. Também podemos mudar o nome do resultado gerado:

mysql> SELECT CONCAT("('", name, "'),") AS "INSERT INTO biblioteca_musical.genero (nome) VALUES " FROM genre WHERE name != '';

Saída gerada:

```
INSERT INTO biblioteca_musical.genero (nome) VALUES
('Alt. Rock'),
('Blues'),
('Brazil'),
('Folk Rock'),
('genre'),
('Hard Rock'),
('Heavy Metal'),
('Karaoké'),
('Latin'),
('Métal'),
('Metal'),
('misc'),
('Miscellaneous'),
('MPB'),
('Other'),
('Pop'),
('pop'),
('Pop/Hair Metal'),
('Pop/Rock'),
('Power Metal'),
('Progressive Rock'),
('Punk Rock'),
('Rap'),
('rock'),
('Rock'),
('Rock & Roll'),
('Rock/Pop'),
('Sound Track'),
('Unknown'),
```

18 – Utilizando vários bancos de dados:

Agora podemos inserir o resultado dessa consulta:

```
mysql> INSERT INTO biblioteca_musical.genero (nome) VALUES
    -> ('Alt. Rock'),
    -> ('Blues'),
    -> ('Brazil'),
    -> ('Folk Rock'),
    -> ('genre'),
    -> ('Hard Rock'),
    -> ('Heavy Metal'),
    -> ('Karaoké'),
    -> ('Latin'),
    -> ('Métal'),
    -> ('Metal'),
    -> ('misc'),
    -> ('Miscellaneous'),
    -> ('MPB'),
    -> ('Other'),
    -> ('Pop'),
    -> ('pop'),
    -> ('Pop/Hair Metal'),
    -> ('Pop/Rock'),
    -> ('Power Metal'),
    -> ('Progressive Rock'),
    -> ('Punk Rock'),
    -> ('Rap'),
    -> ('rock'),
    -> ('Rock'),
    -> ('Rock & Roll'),
    -> ('Rock/Pop'),
    -> ('Sound Track'),
    -> ('Unknown');
```

18 – Utilizando vários bancos de dados:

Outra maneira seria executarmos essa consulta de fora do cliente, utilizando a consulta na linha de comando:

julio@house:~/MySQL$ mysql -p -u root -P 3306 -D amarok2 -e "SELECT name FROM genre WHERE name != ";" > RESULTADO.SQL

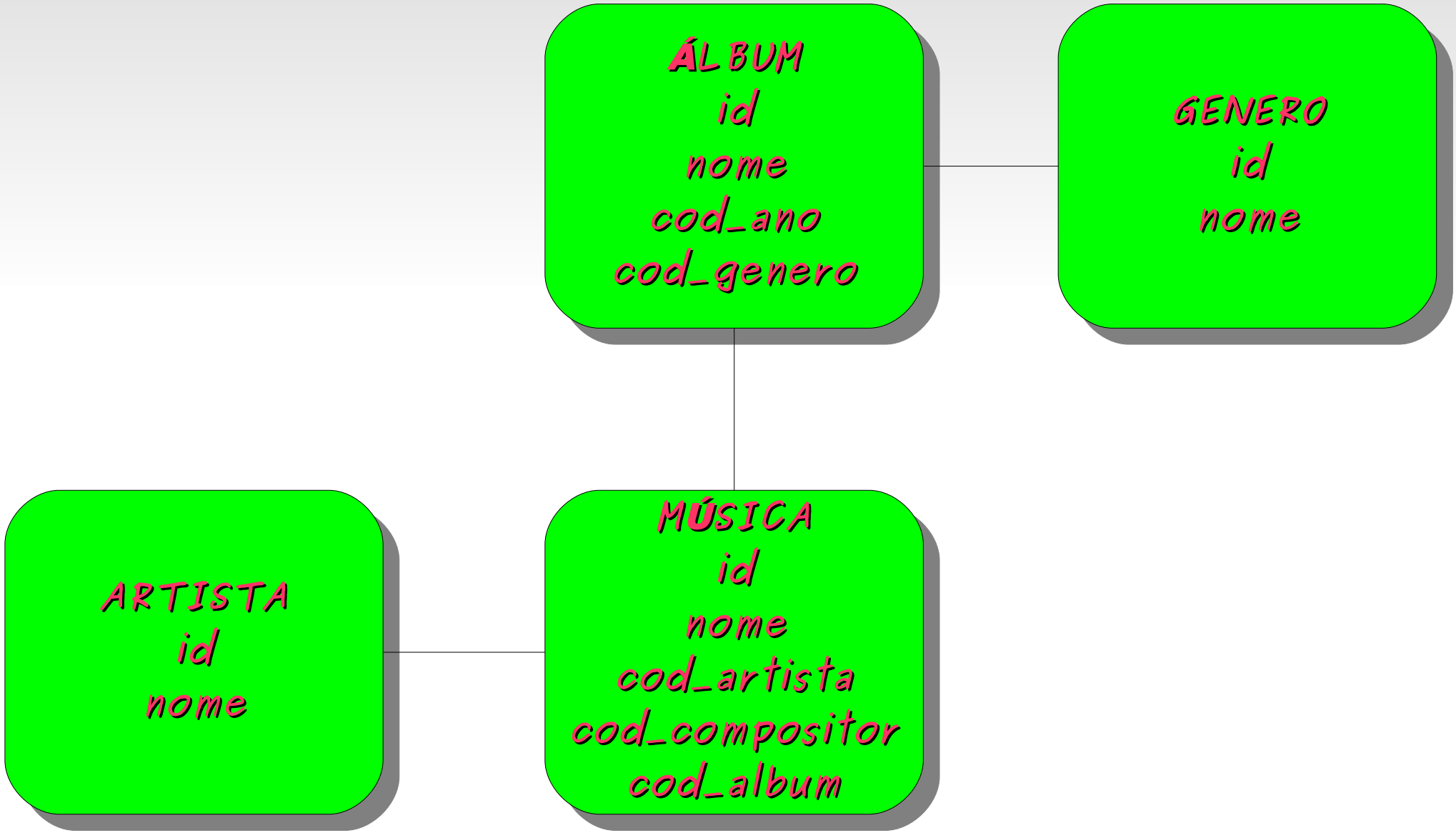
O conteúdo do arquivo é o mesmo da consulta:

```
name
Alt. Rock
Blues
Brazil
Folk Rock
genre
Hard Rock
Heavy Metal
Karaoké
Latin
Métal
Metal
misc
Miscellaneous
MPB
Other
Pop
pop
Pop/Hair Metal
Pop/Rock
Power Metal
Progressive Rock
Punk Rock
Rap
rock
Rock
Rock & Roll
Rock/Pop
Sound Track
Unknown
```

19 Teste 1:

Agora faremos uma utilização mais prática dos conceitos passados até aqui. Faremos consultas e visões mais complexas. Mas ainda assim, passo-a-passo.

Vamos fazer uma consulta que nos retorne o nome da música, o artista e o seu gênero. Vamos ver alguns passos:



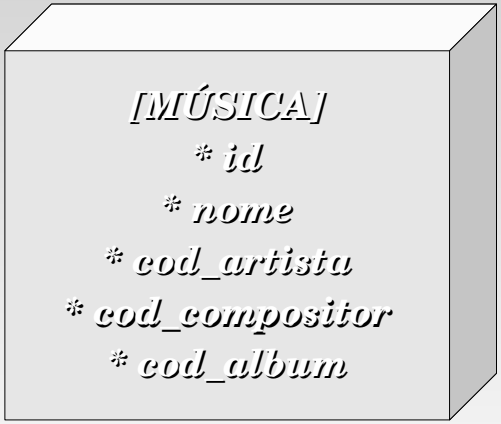
19 Teste 1:

1 - Vamos verificar todas as músicas:

Saída gerada:

```
+----+--------------------------------+-------------+----------------+-----------+
| id | nome                           | cod_artista | cod_compositor | cod_album |
+----+--------------------------------+-------------+----------------+-----------+
|  1 | Cold Winter Nights (Bonus Trac |          26 |           NULL |        14 |
|  2 | SOS                            |          26 |           NULL |        14 |
|  3 | No Turning Back                |          26 |           NULL |        14 |
|  4 | Rebel                          |          26 |           NULL |        14 |
|  5 | Playing with Fire              |          26 |           NULL |        14 |
|  6 | Venus in the Morning           |          26 |           NULL |        14 |
|  7 | 4000 Rainy Nights              |          26 |           NULL |        14 |
|  8 | Anthem of the World            |          26 |           NULL |        14 |
|  9 | Years Go By                    |          26 |           NULL |        14 |
| 10 | Destiny                        |          26 |           NULL |        14 |
+----+--------------------------------+-------------+----------------+-----------+
10 rows in set (0.00 sec)
```

19 Teste 1:



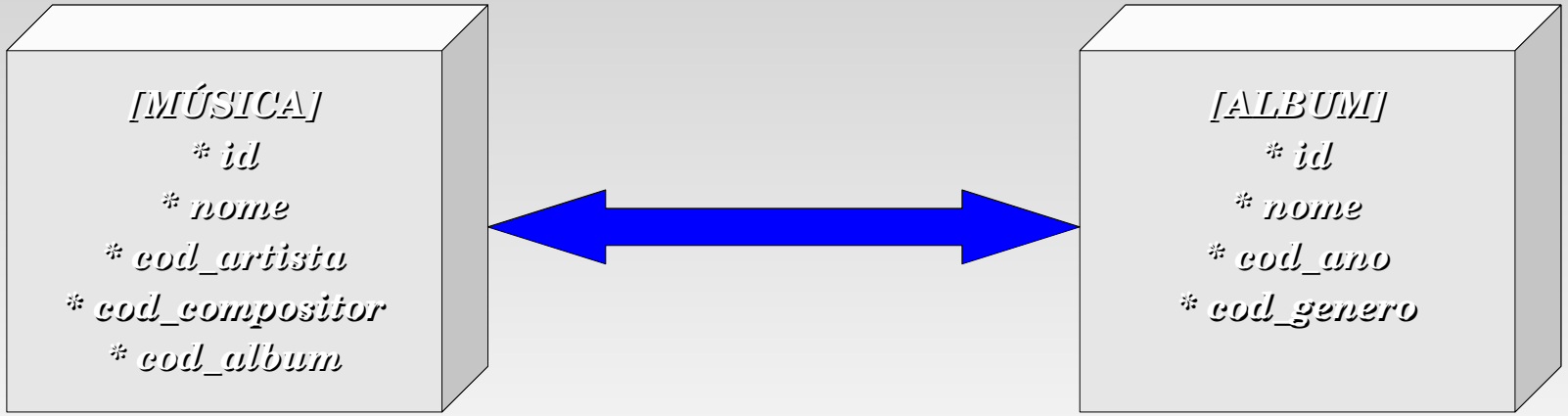
19 Teste 1:

2 - Efetuar um CROSS JOIN entre a tabela musica e album:

Saída gerada:

```
+----+---------------------------------+-------------+----------------+-----------+----+----------------------+---------+------------+
| id | nome                            | cod_artista | cod_compositor | cod_album | id | nome                 | cod_ano | cod_genero |
+----+---------------------------------+-------------+----------------+-----------+----+----------------------+---------+------------+
|  1 | Cold Winter Nights (Bonus Trac |          26 |           NULL |        14 |  1 | Stratovarius         |    NULL |       NULL |
|  2 | SOS                             |          26 |           NULL |        14 |  1 | Stratovarius         |    NULL |       NULL |
|  3 | No Turning Back                 |          26 |           NULL |        14 |  1 | Stratovarius         |    NULL |       NULL |
|  9 | Years Go By                     |          26 |           NULL |        14 | 14 | Destiny              |    1998 |          1 |
| 10 | Destiny                         |          26 |           NULL |        14 | 14 | Destiny              |    1998 |          1 |
|... | ...                             |         ... |            ... |       ... |... | ...                  |     ... |        ... |
|... | ...                             |         ... |            ... |       ... |... | ...                  |     ... |        ... |
|... | ...                             |         ... |            ... |       ... |... | ...                  |     ... |        ... |
|... | ...                             |         ... |            ... |       ... |... | ...                  |     ... |        ... |
|... | ...                             |         ... |            ... |       ... |... | ...                  |     ... |        ... |
|... | ...                             |         ... |            ... |       ... |... | ...                  |     ... |        ... |
|... | ...                             |         ... |            ... |       ... |... | ...                  |     ... |        ... |
|  7 | 4000 Rainy Nights               |          26 |           NULL |        14 | 15 | Fourth Dimension     |    NULL |       NULL |
|  8 | Anthem of the World             |          26 |           NULL |        14 | 15 | Fourth Dimension     |    NULL |       NULL |
|  9 | Years Go By                     |          26 |           NULL |        14 | 15 | Fourth Dimension     |    NULL |       NULL |
| 10 | Destiny                         |          26 |           NULL |        14 | 15 | Fourth Dimension     |    NULL |       NULL |
+----+---------------------------------+-------------+----------------+-----------+----+----------------------+---------+------------+
150 rows in set (0.00 sec)
```

19 Teste 1:


[MÚSICA]
* id
* nome
* cod_artista
* cod_compositor
* cod_album
[ALBUM]
* id
* nome
* cod_ano
* cod_genero

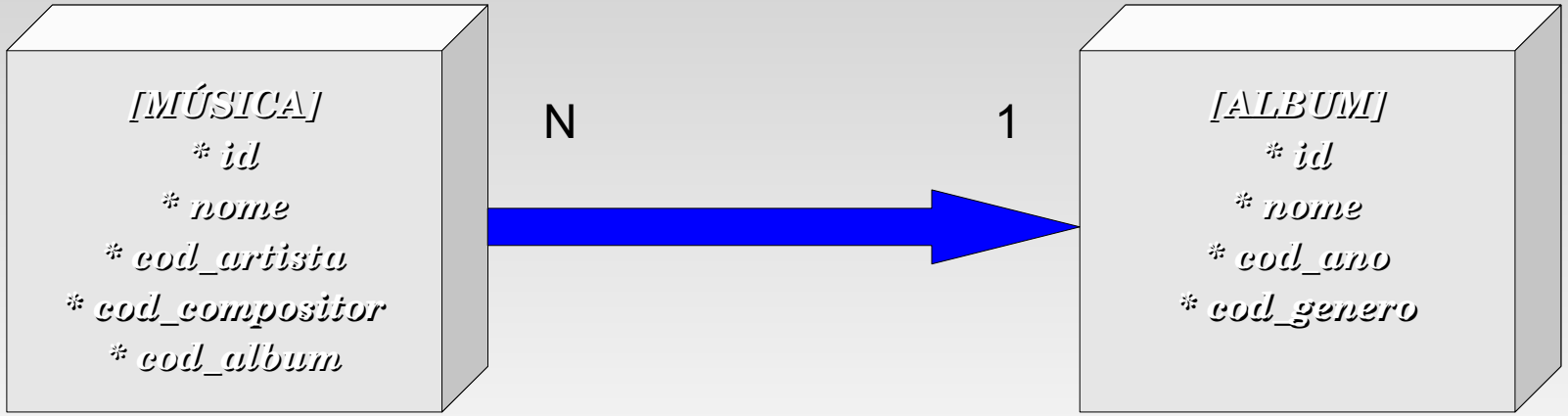
19 Teste 1:

3 - Efetuar o INNER JOIN:

Saída gerada:

```
+----+--------------------------------+-------------+----------------+-----------+----+---------+---------+------------+
| id | nome                           | cod_artista | cod_compositor | cod_album | id | nome    | cod_ano | cod_genero |
+----+--------------------------------+-------------+----------------+-----------+----+---------+---------+------------+
|  1 | Cold Winter Nights (Bonus Trac |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
|  2 | SOS                            |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
|  3 | No Turning Back                |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
|  4 | Rebel                          |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
|  5 | Playing with Fire              |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
|  6 | Venus in the Morning           |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
|  7 | 4000 Rainy Nights              |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
|  8 | Anthem of the World            |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
|  9 | Years Go By                    |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
| 10 | Destiny                        |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |
+----+--------------------------------+-------------+----------------+-----------+----+---------+---------+------------+
10 rows in set (0.00 sec)
```

19 Teste 1:


[MÚSICA]
* id
* nome
* cod_artista
* cod_compositor
* cod_album
N
1
[ALBUM]
* id
* nome
* cod_ano
* cod_genero

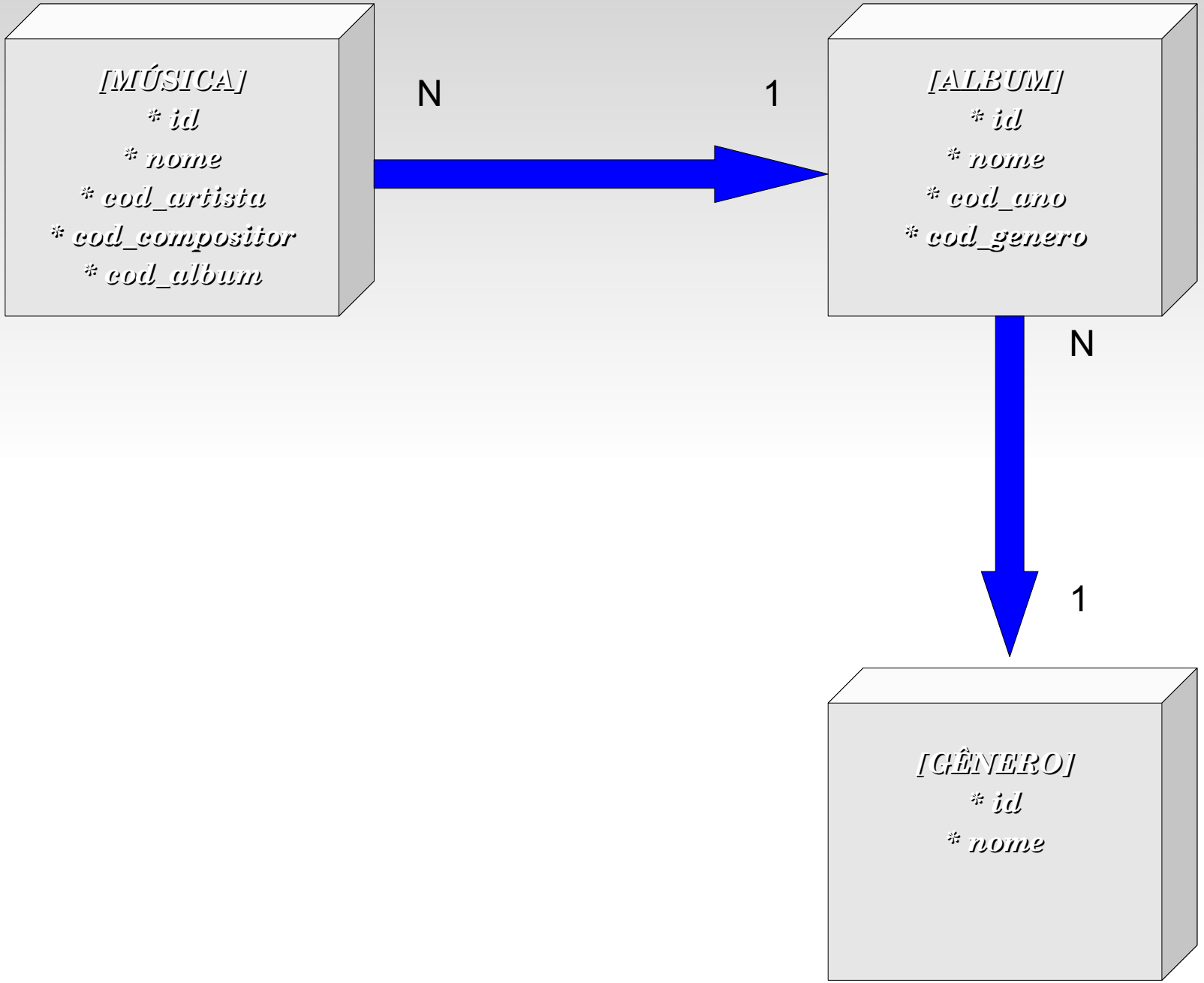
19 Teste 1:

4 - Relacionar o gênero: Agora temos as músicas relacionadas aos seus álbuns. Já podemos verificar o gênero de cada música.

Saída gerada:

```
+----+--------------------------------+-------------+----------------+-----------+----+---------+---------+------------+----+-------------+
| id | nome                           | cod_artista | cod_compositor | cod_album | id | nome    | cod_ano | cod_genero | id | nome        |
+----+--------------------------------+-------------+----------------+-----------+----+---------+---------+------------+----+-------------+
|  1 | Cold Winter Nights (Bonus Trac |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
|  2 | SOS                            |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
|  3 | No Turning Back                |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
|  4 | Rebel                          |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
|  5 | Playing with Fire              |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
|  6 | Venus in the Morning           |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
|  7 | 4000 Rainy Nights              |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
|  8 | Anthem of the World            |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
|  9 | Years Go By                    |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
| 10 | Destiny                        |          26 |           NULL |        14 | 14 | Destiny |    1998 |          1 |  1 | Heavy Metal |
+----+--------------------------------+-------------+----------------+-----------+----+---------+---------+------------+----+-------------+
10 rows in set (0.00 sec)
```

19 Teste 1:


[MÚSICA]
* id
* nome
* cod_artista
* cod_compositor
* cod_album
N
1
[ALBUM]
* id
* nome
* cod_ano
* cod_genero
N
1
[GÊNERO]
* id
* nome

19 Teste 1:

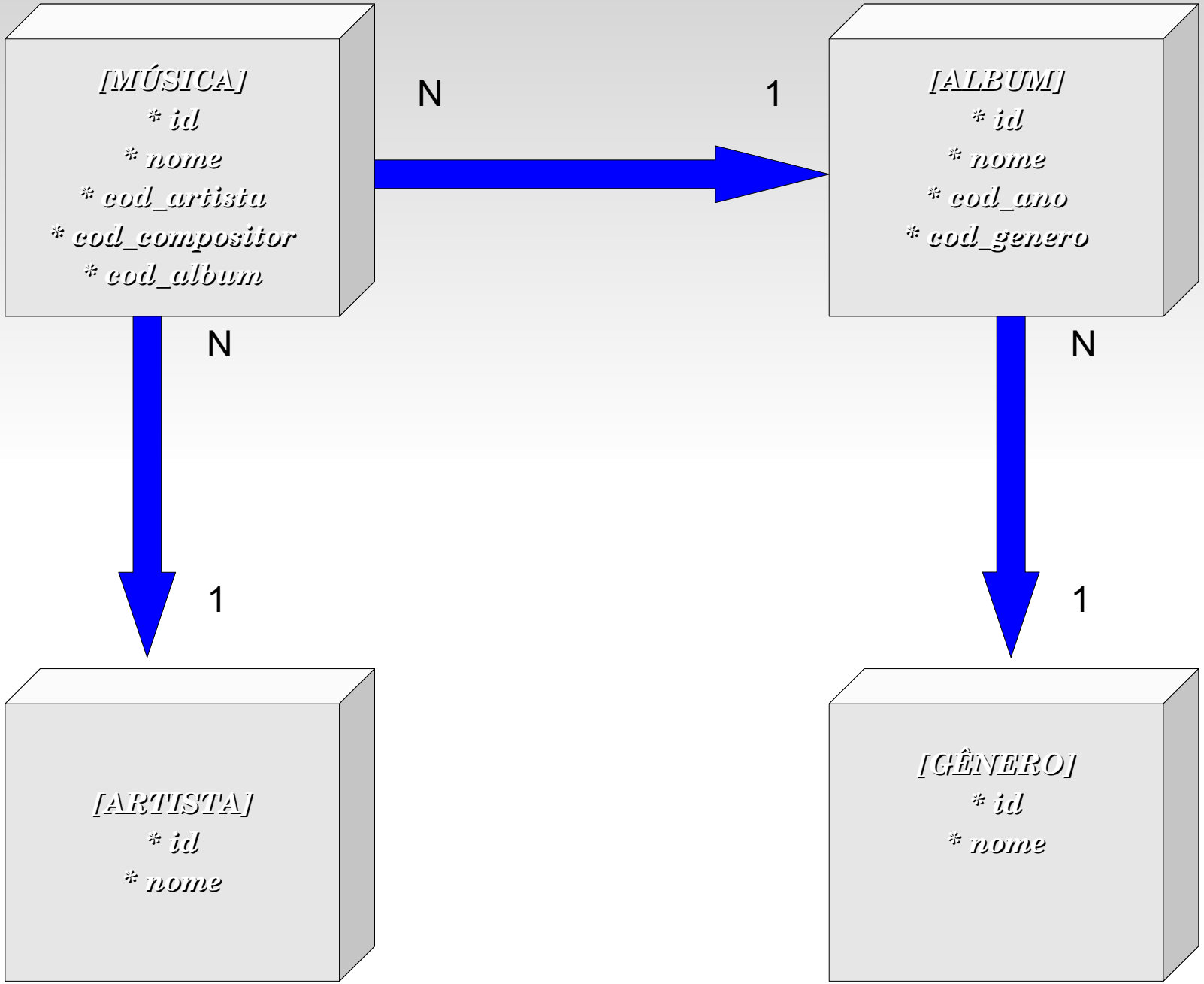
5 - Inserir o artista na consulta: O artista está relacionado à música (artista.id = musica.cod_artista), ligando a tabela, o resultado é o seguinte:

Saída gerada:

id	nome	cod_artista	cod_compositor	cod_album	id	nome	cod_ano	cod_genero	id	nome	id	nome
1	Cold Winter Nights (Bonus Trac	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
2	SOS	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
3	No Turning Back	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
4	Rebel	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
5	Playing with Fire	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
6	Venus in the Morning	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
7	4000 Rainy Nights	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
8	Anthem of the World	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
9	Years Go By	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius
10	Destiny	26	NULL	14	14	Destiny	1998	1	1	Heavy Metal	26	Stratovarius

10 rows in set (0.00 sec)

19 Teste 1:


[MÚSICA]
* id
* nome
* cod_artista
* cod_compositor
* cod_album
N
1
[ALBUM]
* id
* nome
* cod_ano
* cod_genero
N
1
N
1
[ARTISTA]
* id
* nome
[GÊNERO]
* id
* nome

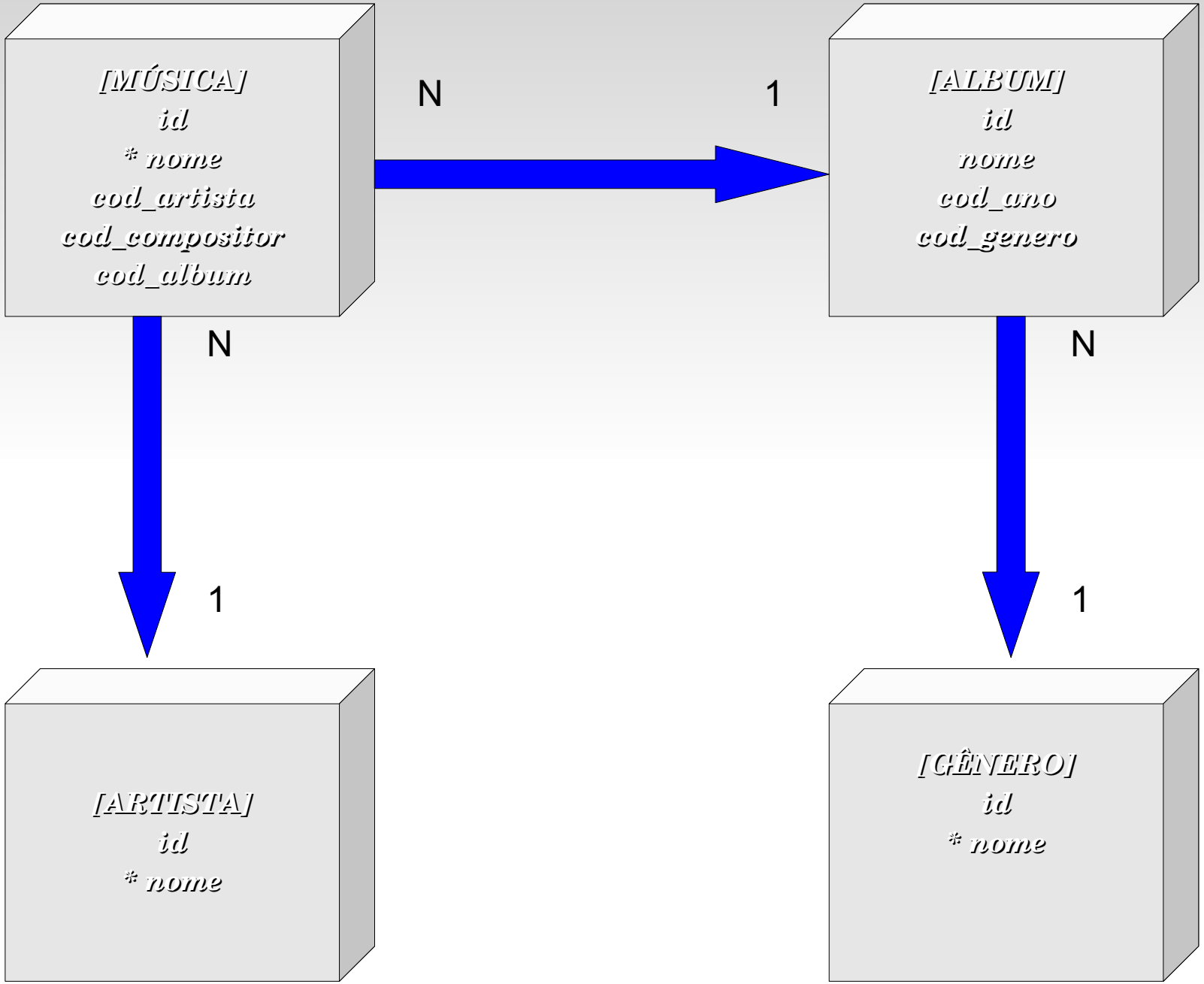
19 Teste 1:

6 - Purificando o resultado: Queremos visualizar apenas o nome da música, o artista e o seu gênero:

Saída gerada:

```
+--------------------------------+--------------+-------------+
| Musica                         | Artista      | Genero      |
+--------------------------------+--------------+-------------+
| Cold Winter Nights (Bonus Trac | Stratovarius | Heavy Metal |
| SOS                            | Stratovarius | Heavy Metal |
| No Turning Back                | Stratovarius | Heavy Metal |
| Rebel                          | Stratovarius | Heavy Metal |
| Playing with Fire              | Stratovarius | Heavy Metal |
| Venus in the Morning           | Stratovarius | Heavy Metal |
| 4000 Rainy Nights              | Stratovarius | Heavy Metal |
| Anthem of the World            | Stratovarius | Heavy Metal |
| Years Go By                    | Stratovarius | Heavy Metal |
| Destiny                        | Stratovarius | Heavy Metal |
+--------------------------------+--------------+-------------+
10 rows in set (0.00 sec)
```

19 Teste 1:


[MÚSICA]
id
* nome
cod_artista
cod_compositor
cod_album
N
1
[ALBUM]
id
nome
cod_ano
cod_genero
N
1
N
1
[ARTISTA]
id
* nome
[GÊNERO]
id
* nome

19 Teste 1:

7 - Montar a visão para facilitar nossas consultas (vamos chamar a visão de mus_art_gen):

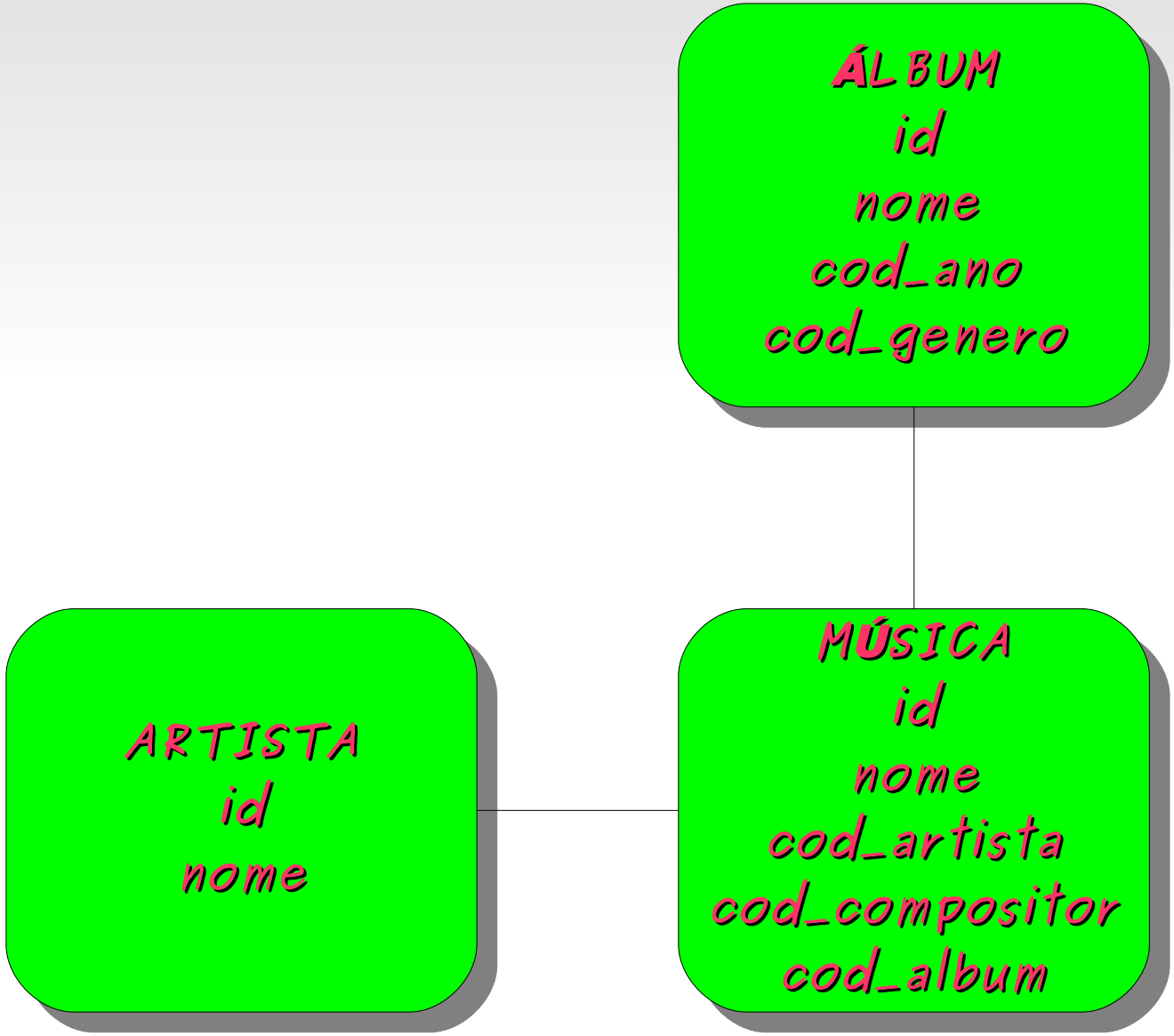
mysql> SELECT * FROM mus_art_gen;

Saída gerada:

```
+--------------------------------+--------------+-------------+
| Musica                         | Artista      | Genero      |
+--------------------------------+--------------+-------------+
| Cold Winter Nights (Bonus Trac | Stratovarius | Heavy Metal |
| SOS                            | Stratovarius | Heavy Metal |
| No Turning Back                | Stratovarius | Heavy Metal |
| Rebel                          | Stratovarius | Heavy Metal |
| Playing with Fire              | Stratovarius | Heavy Metal |
| Venus in the Morning           | Stratovarius | Heavy Metal |
| 4000 Rainy Nights              | Stratovarius | Heavy Metal |
| Anthem of the World            | Stratovarius | Heavy Metal |
| Years Go By                    | Stratovarius | Heavy Metal |
| Destiny                        | Stratovarius | Heavy Metal |
+--------------------------------+--------------+-------------+
10 rows in set (0.00 sec)
```

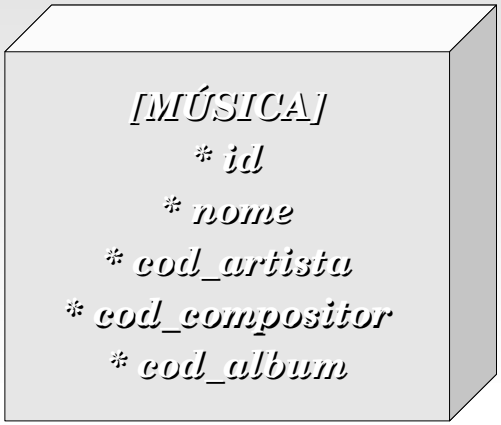
20 Teste 2:

A consulta a seguir é mais simples de realizar. Vamos mostrar apenas o nome do álbum, do artista e da música.



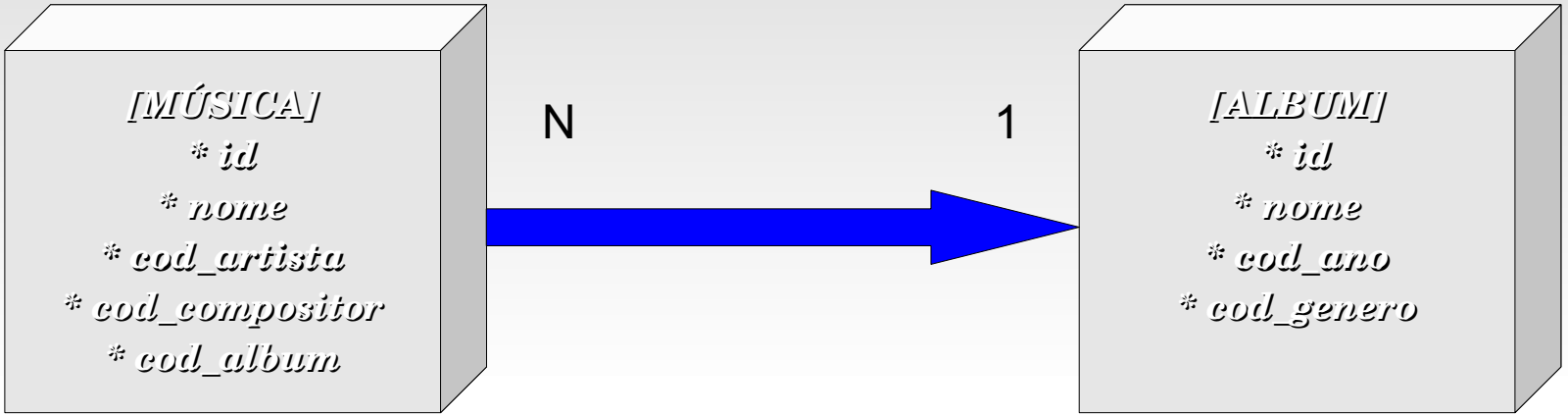
20 Teste 2:

1 - Começamos pela tabela de músicas:



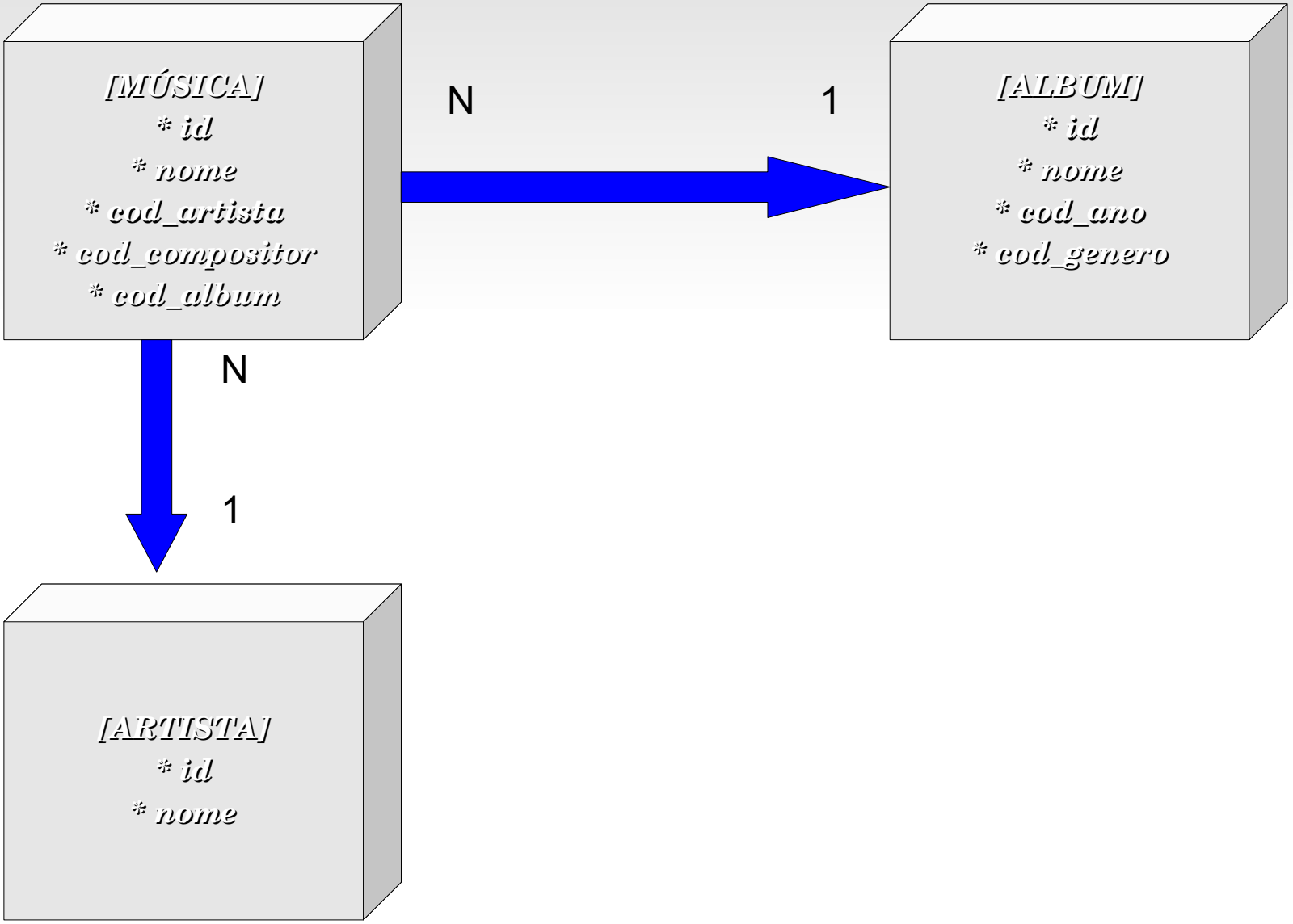
20 Teste 2:

2 - Após, faremos um INNER JOIN com a tabela de álbum:



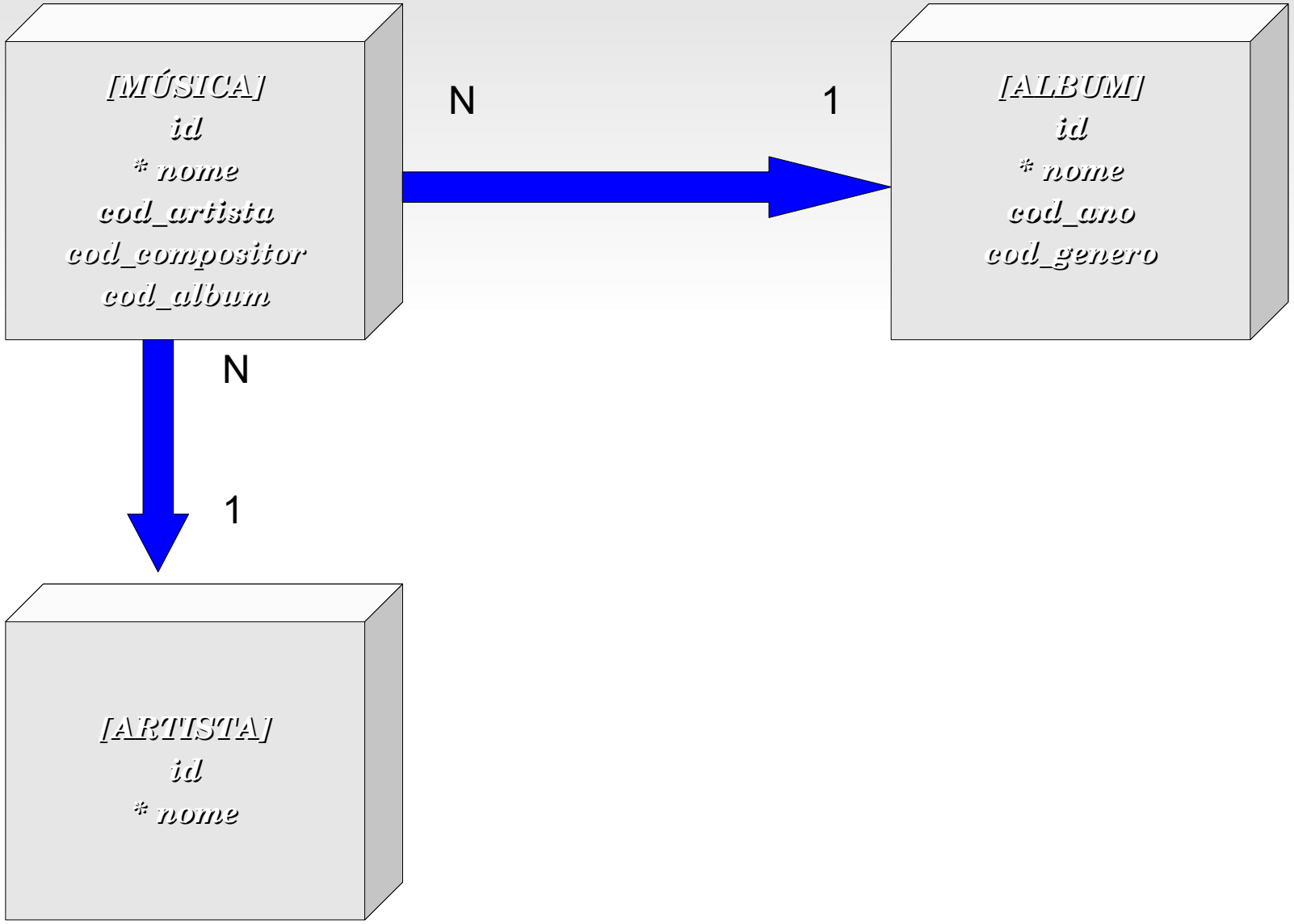
20 Teste 2:

3 - E terminamos fazendo um INNER JOIN de nossa última seleção com a tabela de artistas:



20 Teste 2:

4 - Selecionamos os campos necessários:



20 Teste 2:

5 - Criar a VIEW:

Saída gerada:

```
mysql> SELECT * FROM alb_art_mus;
+---------+-------------+--------------------------------+
| Album   | Artista     | Musica                         |
+---------+-------------+--------------------------------+
| Destiny | Stratovarius | Cold Winter Nights (Bonus Trac |
| Destiny | Stratovarius | SOS                            |
| Destiny | Stratovarius | No Turning Back                |
| Destiny | Stratovarius | Rebel                          |
| Destiny | Stratovarius | Playing with Fire              |
| Destiny | Stratovarius | Venus in the Morning           |
| Destiny | Stratovarius | 4000 Rainy Nights              |
| Destiny | Stratovarius | Anthem of the World            |
| Destiny | Stratovarius | Years Go By                    |
| Destiny | Stratovarius | Destiny                        |
+---------+-------------+--------------------------------+
10 rows in set (0.02 sec)
```

21 Commit e rollback:

Para o uso empresarial de um banco de dados é necessário que este apresente um sistema de recuperação de transação para casos de falhas. Um banco de dados deve ser capaz de retornar a um determinado estado em caso de erro. Desde a versão 3.23 do MySQL este recurso está inserido.
Vamos analisar o exemplo mais clássico: O das contas bancárias.
Uma função em determinada linguagem de programação efetua a seguinte troca de saldos entre as contas: Mover 300 reais da conta 1 para a 2:

```
int id1, id2;
float troca;
...
id1 = 111; id2 = 222;
troca = 300;
sql = "
UPDATE conta SET conta.saldo = conta.saldo - " + troca + " WHERE conta.id = " + id1 + ";
UPDATE conta SET conta.saldo = conta.saldo + " + troca + " WHERE conta.id = " + id2 + ";
"
```

A execução desse código gera a seguinte string sql:

```
UPDATE conta SET conta.saldo = conta.saldo - 300 WHERE conta.id = 111;
UPDATE conta SET conta.saldo = conta.saldo + 300 WHERE conta.id = 222;
```

21 Commit e rollback:

Este bloco de consulta se executado até o seu final, atualizará as duas contas. Uma com trezentos reais a menos e outra com trezentos reais a mais. Tudo estará certo. Mas se algum problema acontecer no servidor (falta de energia, falha na rede, etc) e apenas a primeira parte do bloco for realizada, teremos uma diferença de R$ 300,00 nos saldos. Uma conta teve seu débito efetuado, mas a segunda não teve se crédito realizado. O sistema terá perdido R$ 300,00 da segunda conta. Para evitarmos tais problemas temos o controle de transações. O controle de transações garante que todas as operações serão realizadas, ou em caso negativo, nenhuma acontecerá (na verdade, todas são desativadas). Isso garante que no nosso exemplo, ou as duas contas terão seu saldo atualizado, ou nenhuma será alterada.
No MySQL este controle pode ser efetuado para tipos de tabelas INNODB.
Existe um parâmetro no banco que controla as transações. Este parâmetro se chama AUTOCOMMIT. Por padrão ele vem ativado. Isto significa que cada atualização no banco será executada realmente. Após um comando, o COMMIT é executado sem que o usuário do banco perceba. Dessa forma qualquer alteração no banco de dados será gravada automaticamente. Mas caso seja necessário o controle de algumas tarefas (por exemplo, somente validar a transação se as duas contas forem alteradas), podemos utilizar os comandos de início de transação (START TRANSACTION) e gravação de transação (COMMIT). Em caso de erro durante o processo, um comando ROLLBACK pode ser executado para voltar o banco ao estado anterior ao START TRANSACTION.
Após o início de uma transação, já podemos efetuar nossas instruções de sql (imaginemos as duas instruções, subtrair o saldo de uma conta e adicionar na outra). Após todas as instruções realizadas com sucesso, podemos confirmá-las e realmente ativá-las. Confirmamos com o COMMIT. Caso um erro ocorra e seja necessário retornarmos a um estado anterior, efetuamos o processo de ROLLBACK.
Então para que nossas instruções ocorram do início ao fim, podemos definir a seguinte estrutura:

```
START TRANSACTION;
[INSTRUÇÕES]
COMMIT;
```

21 Commit e rollback:

Após o COMMIT os dados são gravados. Mas se houvesse um problema e apenas a primeira instrução fosse realizada, nosso banco não estaria conciso. Uma consulta ao banco de dados nesse momento poderia mostrar dados incorretos. Mas ao efetuarmos um ROLLBACK nosso banco volta ao estado em que se encontrava antes do START TRANSACTION.
Vamos inserir alguns dados para podermos testar:

mysql> CREATE DATABASE banco_comercial;

Saída gerada:
```
Query OK, 1 row affected (0.04 sec)
```

mysql> use banco_comercial

Saída gerada:
```
Database changed
```

mysql> CREATE TABLE conta (id int primary key auto_increment, saldo float, nome text) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.03 sec)
```

mysql> INSERT INTO conta (nome, saldo) VALUES ('Tania', 30.50), ('Aparecida', 50.80);

Saída gerada:
```
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

21 Commit e rollback:

Temos duas contas:

mysql> SELECT CONCAT("R$: ", saldo) AS Saldo, nome AS Nome FROM conta;

Saída gerada:

```
+----------+-----------+
| Saldo    | Nome      |
+----------+-----------+
| R$: 30.5 | Tania     |
| R$: 50.8 | Aparecida |
+----------+-----------+
2 rows in set (0.05 sec)
```

Agora podemos fazer nossas trocas de saldo. Utilizando uma transação, podemos ver os saldos atualizados. Mas caso não ocorra tudo como esperado e um ROLLBACK seja executado, estes saldos voltarão a seu estado original. Vamos agora observar dois exemplos: O primeiro tudo aconteceu como previsto, R$ 50,00 foram transferidos. mas no segundo uma pane ocorreu ao transferirmos R$ 20,00. Vamos verificar o total das contas bancárias com o comando SUM, que faz um somatório de alguma coluna:

mysql> SELECT SUM(saldo) FROM conta;

Saída gerada:

```
+------------+
| SUM(saldo) |
+------------+
|       81.3 |
+------------+
1 row in set (0.00 sec)
```

21 Commit e rollback:

PRIMEIRO EXEMPLO:

mysql> START TRANSACTION;

Saída gerada:

```
Query OK, 0 rows affected (0.02 sec)
```

Atualizando a primeira conta:

mysql> UPDATE conta SET saldo = saldo + 50 WHERE id = 1;

Saída gerada:

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

O resultado é visível, mas caso ocorresse um ROLLBACK ele seria revertido:

mysql> SELECT * FROM conta;

Saída gerada:

```
+----+-------+-----------+
| id | saldo | nome      |
+----+-------+-----------+
|  1 |  80.5 | Tania     |
|  2 |  50.8 | Aparecida |
+----+-------+-----------+
2 rows in set (0.00 sec)
```

21 Commit e rollback:

Atualizando a segunda conta:

mysql> UPDATE conta SET saldo = saldo - 50 WHERE id = 2;

Saída gerada:

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Confirmando a realização com sucesso da transação:

mysql> COMMIT;

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

Verificando o saldo:

mysql> SELECT * FROM conta;

Saída gerada:

```
+----+-------+-----------+
| id | saldo | nome      |
+----+-------+-----------+
|  1 |  80.5 | Tania     |
|  2 |  0.80 | Aparecida |
+----+-------+-----------+
2 rows in set (0.00 sec)
```

21 Commit e rollback:

Verificando o total do saldo do banco:

mysql> SELECT SUM(saldo) FROM conta;

Saída gerada:

```
+-------------+
| SUM(saldo)  |
+-------------+
|        81.3 |
+-------------+
1 row in set (0.00 sec)
```

SEGUNDO EXEMPLO:

mysql> START TRANSACTION;

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

Agora atualizando a primeira conta:

mysql> UPDATE conta SET saldo = saldo - 20 WHERE id = 1;

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

21 Commit e rollback:

Saldo até o momento:

mysql> SELECT * FROM conta;

Saída gerada:

```
+----+-------+-----------+
| id | saldo | nome      |
+----+-------+-----------+
|  1 |  60.5 | Tania     |
|  2 |   0.8 | Aparecida |
+----+-------+-----------+
2 rows in set (0.00 sec)
```

Se nesse momento um problema ocorrer e nossa segunda parte da transação não for executada, nosso banco estaria com as tabelas conta incorreto.

mysql> SELECT SUM(saldo) FROM conta;

Saída gerada:

```
+------------+
| SUM(saldo) |
+------------+
|       61.3 |
+------------+
1 row in set (0.01 sec)
```

21 Commit e rollback:

Aqui temos as tabelas de conta, em que faltam R$ 20,00. Para retornarmos ao estado original, podemos utilizar um ROOLBACK:

mysql> ROlLBACK;

Saída gerada:

```
Query OK, 0 rows affected (0.02 sec)
```

Agora voltamos ao estado original:

mysql> SELECT SUM(saldo) FROM conta;

Saída gerada:

```
+------------+
| SUM(saldo) |
+------------+
|       81.3 |
+------------+
1 row in set (0.00 sec)
```

Os saldos voltaram ao estado inicial, antes do START TRANSACTION:

mysql> SELECT * FROM conta;

Saída gerada:

```
+----+-------+-----------+
| id | saldo | nome      |
+----+-------+-----------+
|  1 |  80.5 | Tania     |
|  2 |   0.8 | Aparecida |
+----+-------+-----------+
2 rows in set (0.00 sec)
```

21 Commit e rollback:

Para desativarmos o uso do AUTOCOMMIT podemos setar a variável como 0 (zero):

mysql> SET AUTOCOMMIT=0;

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
```

Agora não precisamos iniciar uma transação. Mas devemos setar sempre um COMMIT para que um comando ROLLBACK não nos faça perdermos dados.
Ainda temos outro recurso interessante: Utilizar pontos de salvamento. Dentro de uma transação podemos voltar até determinado ponto. Utilizamos o comando SAVEPOINT:

SAVEPOINT ponto1

Para retornarmos ao ponto ponto1, utilizamos um ROLLBACK especificando o ponto:

ROLLBACK TO SAVEPOINT ponto1

Durante a gravação dos savepoints uma gravação com mesmo nome, apaga a anterior. E o ponteiro avança.

22 Relacionamento entre as tabelas:

Criaremos agora um sistema que abordará melhor o uso de relacionamentos entre as tabelas. Utilizaremos os conceitos vistos até o momento e também alguns recursos do MySQL ainda não vistos. Em um banco de dados relacional (que efetua relacionamento entre as tabelas), podemos ter três situações durante o projeto do sistema:

-->> 1) Relacionamento UM para VÁRIOS (1 - *): Este é o mais simples de se compreender. No primeiro exemplo criado, temos as tabelas cursos e alunos. Se pensarmos que um aluno pode fazer apenas um curso durante a Jornada de Atualização Tecnológica, podemos supor que um aluno estará relacionado a apenas um curso. Já cada curso está relacionado a vários (um ou mais) alunos. Como o aluno só participará de um curso, na tabela aluno, setamos seu curso (através do uso de uma chave estrangeira).

22 Relacionamento entre as tabelas:

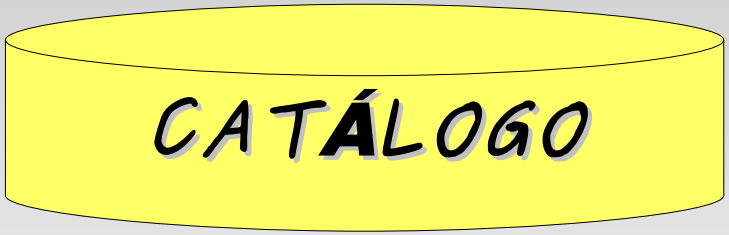
-->> 2) Relacionamento Um para Um (1 - 1): Este relacionamento é menos comum. Quando os elementos de uma tabela se relaciona com apenas um elemento de outra tabela. Embora não utilizamos ainda esse relacionamento, podemos analizar o seguinte exemplo: Em cadastros de pessoas, podemos ter uma tabela para armazenar o RG e outra para armazenar o CPF. Como não podemos ter esses números repetidos, todos tem um CPF e um RG distintos dos demais. A tabela de CPF não fará relação com mais de uma RG e o contrario também. Neste caso o correto a fazer, é unir as tabelas em uma só, com todos os seus campos preservados. Outra alternativa é criar duas chaves estrangeiras (uma em cada tabela) e fazer uma união dupla.

-->> 3) Relacionamento VÁRIOS para VÁRIOS (* - *): Este é o mais interessante. Será estudado adiante. Neste caso, uma tabela faz relação com vários elementos de outra e vice versa. O exemplo estudado será o seguinte: Estamos gravando em um banco, uma coleção de filmes e os seus discos. Se supormos que existem filmes que ficam em vários discos (um ou mais) e que também temos em apenas um disco vários filmes (em um DVD por exemplo), temos uma relação de vários para vários (ou seja, um ou mais para um ou mais). Aqui devemos criar uma tabela intermediária entre as tabelas em questão. Nessa nova tabela faremos as ligações necessárias. Ela terá obrigatoriamente duas chaves estrangeiras (uma para cada tabela) e poderá ter uma chave primária, para garantir a integridade (esta chave pode ser composta).

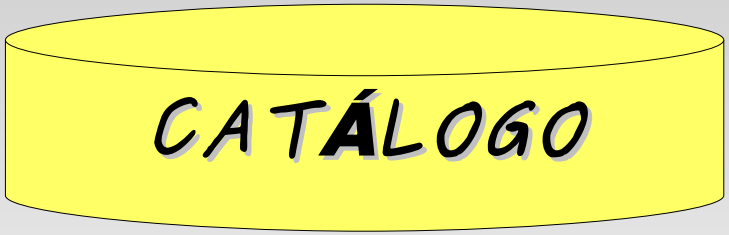
23 Catálogo de filmes:

Agora já temos condições de projetar nosso catálogo de filmes. Teremos tabelas para arquivar desde as produtoras do filme até a marca do CD/DVD. O modelo do banco é o seguinte:
Arquivaremos os filmes em discos. Os discos ficam guardados dentro de estojos. Os discos e os estojos serão numerados.
-->> Um disco pode ter mais de um filme. Da mesma forma, um filme pode estar mais mais de um disco.
-->> Cada filme terá vários atributos (nome, ano, gênero, diretor, atores, produtora, ect), tipo (se é um filme, um seriado, um anime), além das especificações técnicas (altura e largura do vídeo, FPS, qualidade do som, canais de som, codecs, etc).
-->> Deverão existir tabelas para os diretores, atores, produtoras, gêneros.
-->> Os discos terão várias propriedades: Marca, tamanho (700MB, 4.7GB, 9GB, etc).

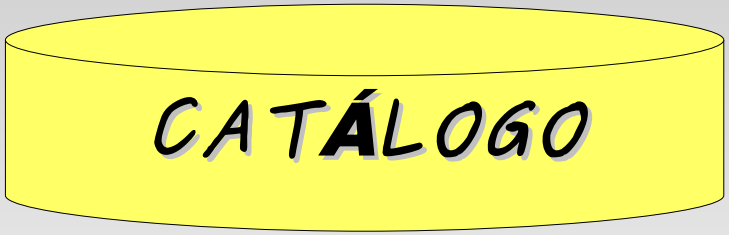
23 Catálogo de filmes:



23 Catálogo de filmes:



23 Catálogo de filmes:

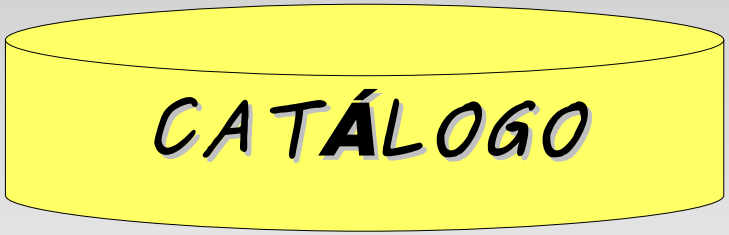


VIDEO

FILME_DISCO

DISCO

23 Catálogo de filmes:



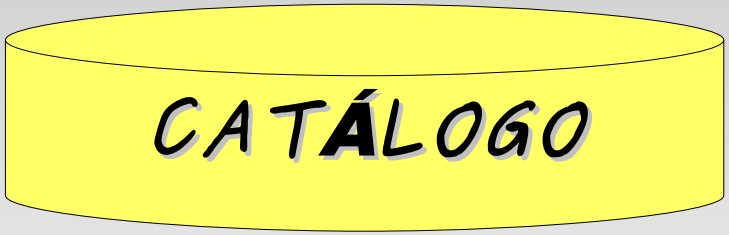
VIDEO

FILME_DISCO

DISCO

MARCA

23 Catálogo de filmes:



VIDEO

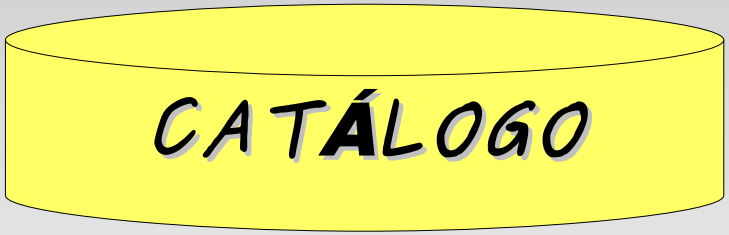
FILME_DISCO

DISCO

MARCA

TAMANHO

23 Catálogo de filmes:



VIDEO

FILME_DISCO

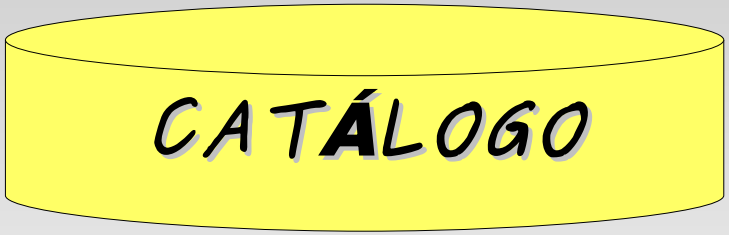
DISCO

MARCA

TAMANHO

ESTOJO

23 Catálogo de filmes:



VIDEO

FILME_DISCO

DISCO

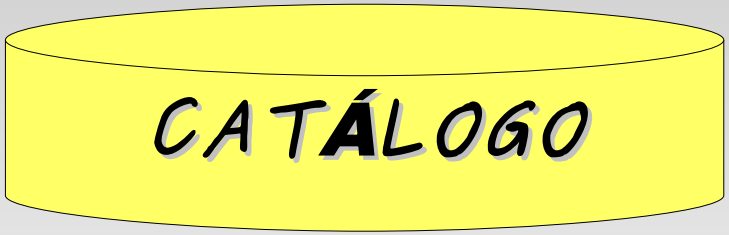
GENERO

MARCA

TAMANHO

ESTOJO

23 Catálogo de filmes:



VIDEO

FILME_DISCO

DISCO

GENERO

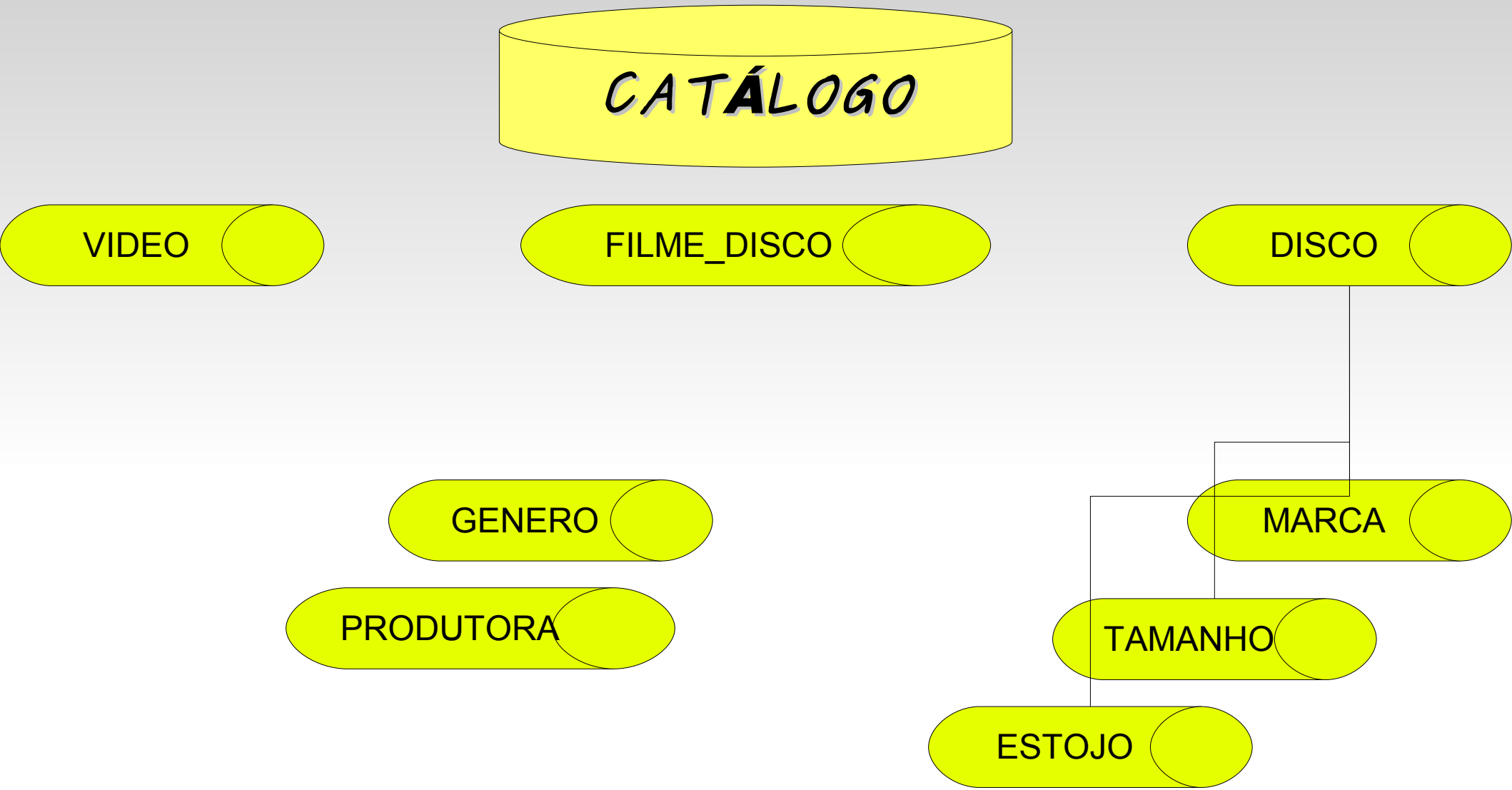
MARCA

PRODUTORA

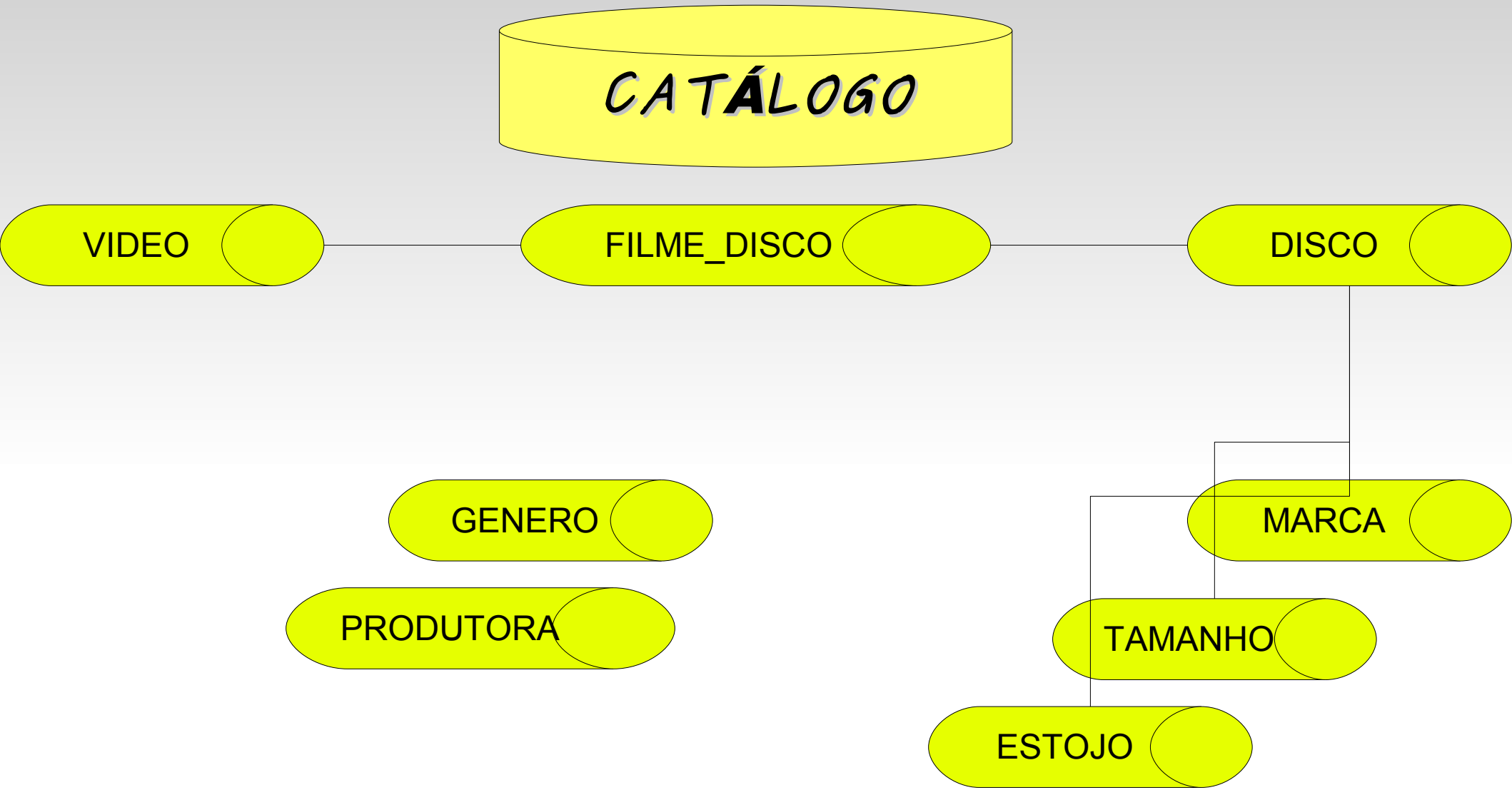
TAMANHO

ESTOJO

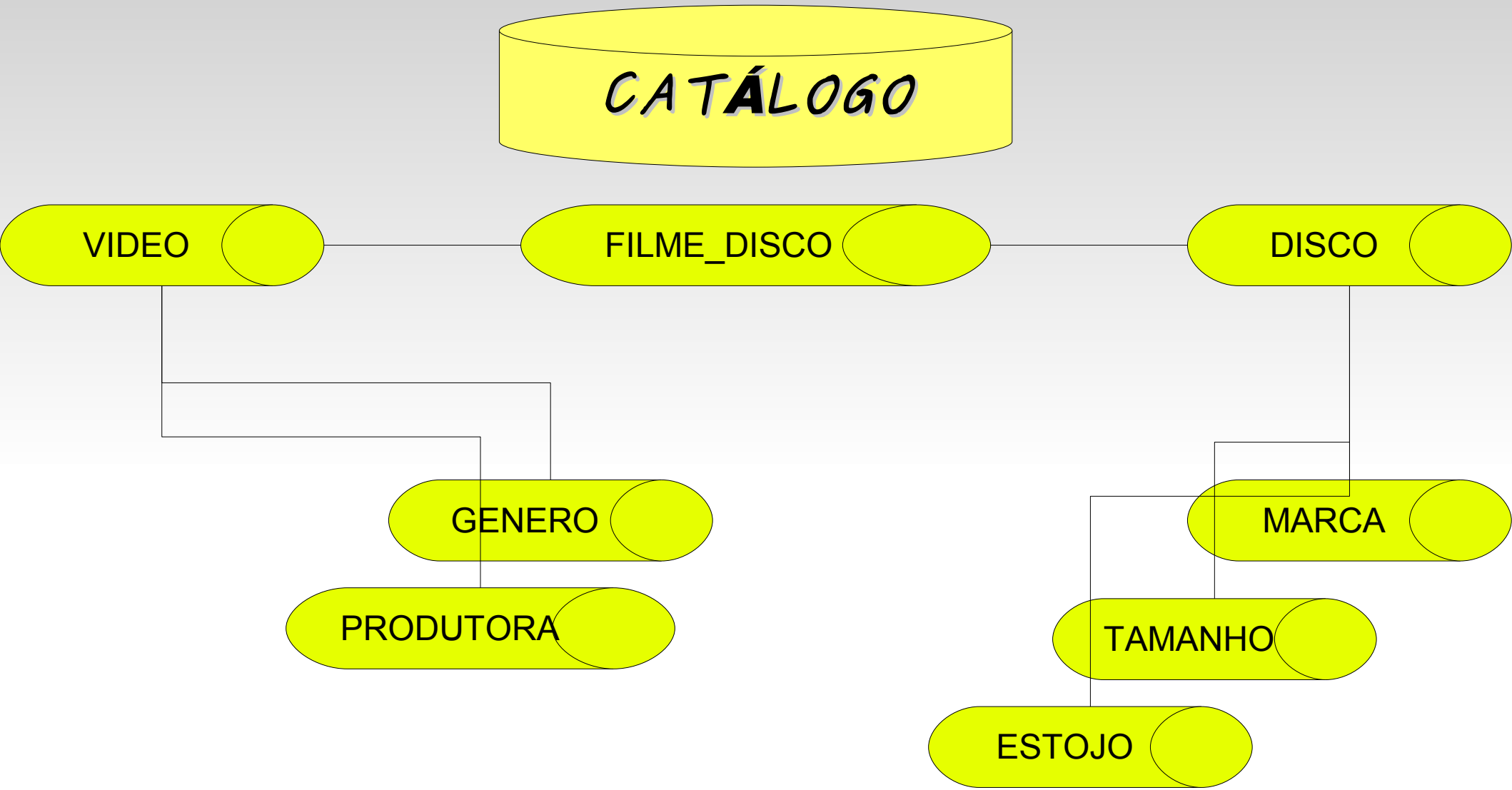
23 Catálogo de filmes:



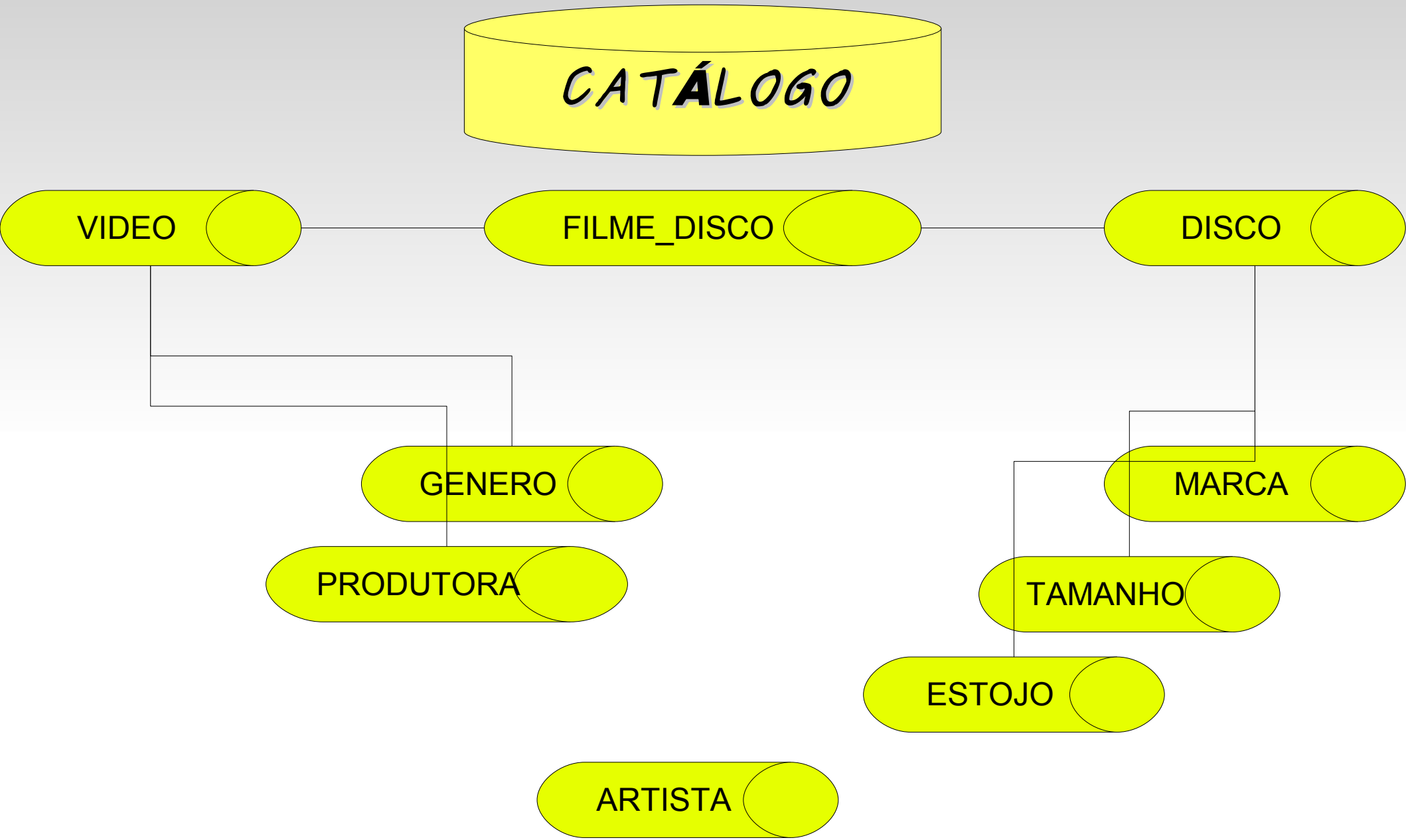
23 Catálogo de filmes:



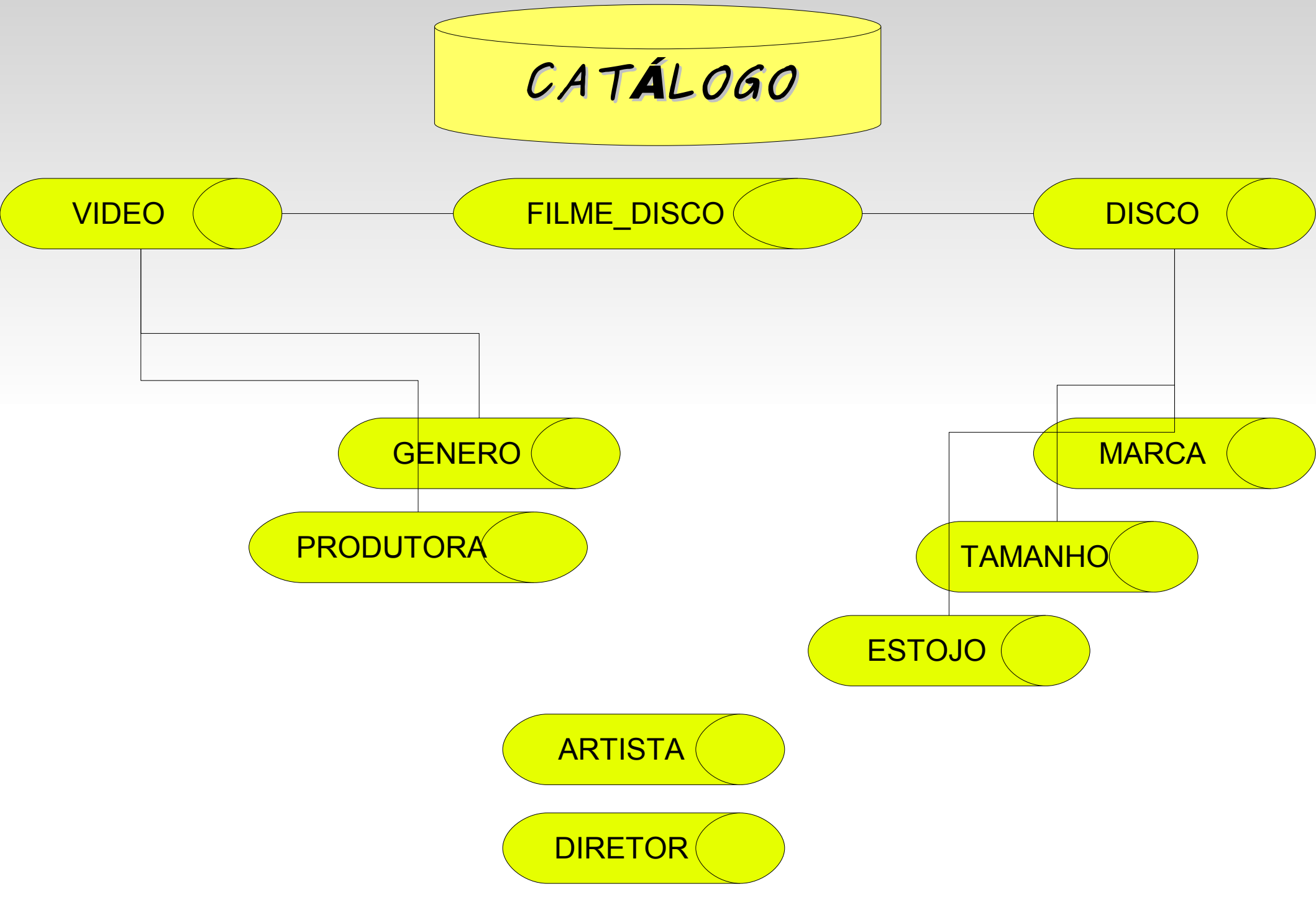
23 Catálogo de filmes:



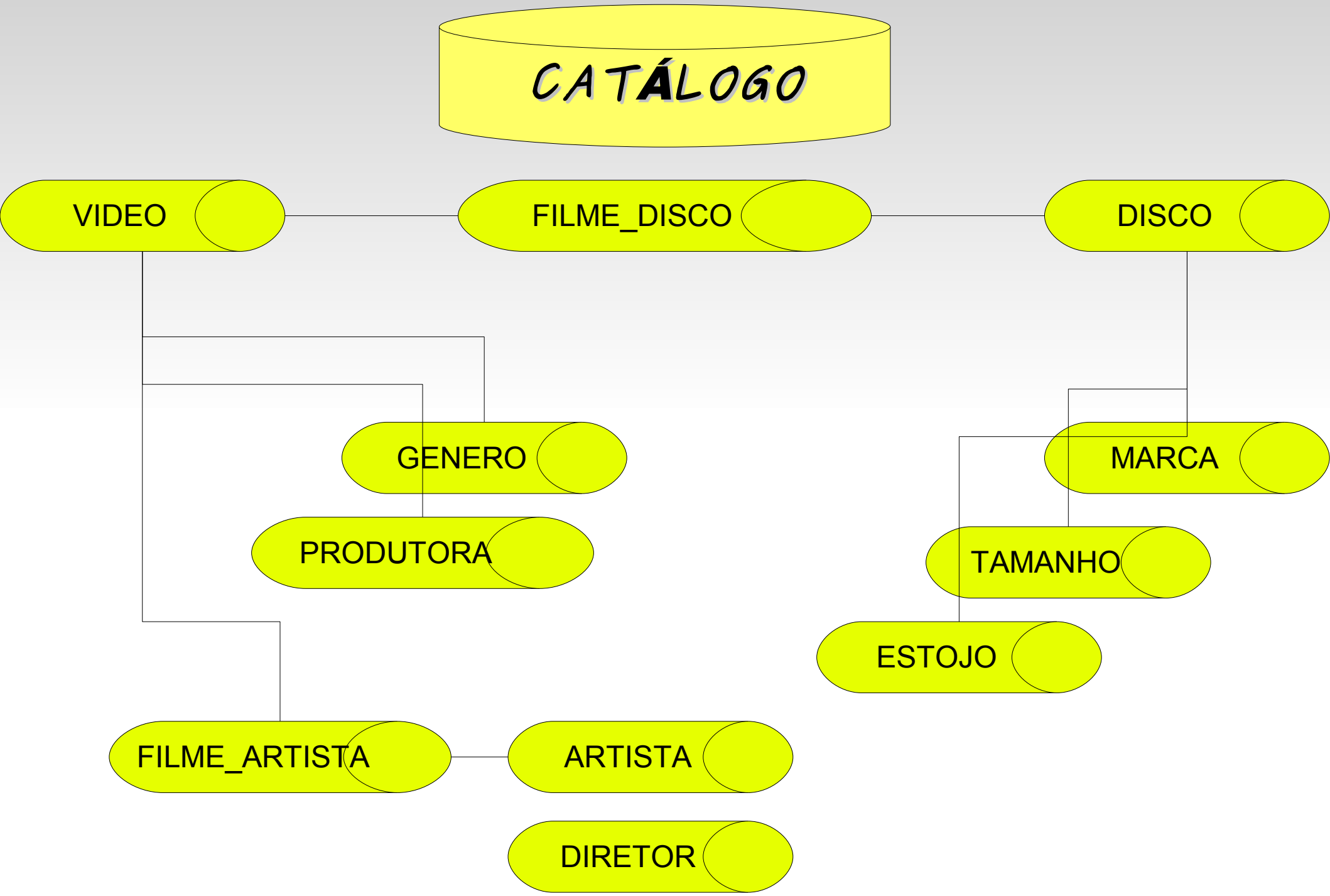
23 Catálogo de filmes:



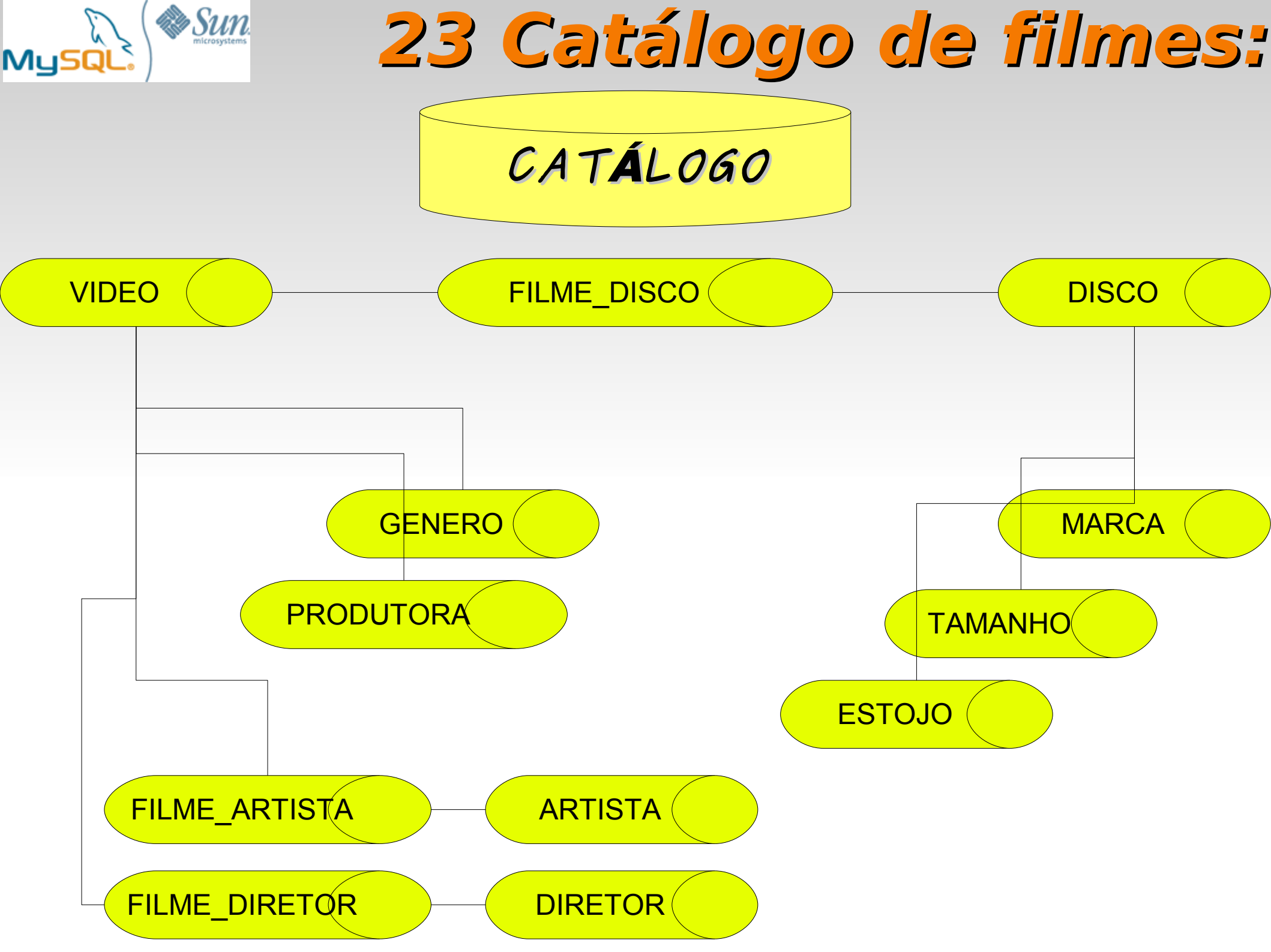
23 Catálogo de filmes:



23 Catálogo de filmes:



23 Catálogo de filmes:



23 Catálogo de filmes:

mysql> CREATE DATABASE catalogo;
```
Query OK, 1 row affected (0.10 sec)
```

mysql> use catalogo
```
Database changed
```

mysql> CREATE TABLE video (id int, nome text, ano int unsigned, altura int unsigned, largura int unsigned, fps int unsigned, codec_img int unsigned, qualidade_som int unsigned, canais_som int unsigned, codec_som int unsigned, cod_genero int, cod_produtora int, cod_tipo int, cod_filme_dir int, cod_filme_atores int, cod_filme_discos int) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.05 sec)
```

mysql> CREATE TABLE disco (id int, cod_marca int, cod_tamanho int, cod_estojo int, bom_estado bool default false) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.05 sec)
```

mysql> CREATE TABLE filme_disco (id int, cod_filme int, cod_disco int) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.01 sec)
```

23 Catálogo de filmes:

mysql> CREATE TABLE marca (id int, nome text) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.01 sec)
```

mysql> CREATE TABLE tamanho (id int, descricao float) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.01 sec)
```

mysql> CREATE TABLE estojo (id int) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.00 sec)
```

mysql> CREATE TABLE genero (id int, nome text) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.00 sec)
```

mysql> CREATE TABLE produtora (id int, nome text) ENGINE = INNODB;

Saída gerada:
```
Query OK, 0 rows affected (0.00 sec)
```

23 Catálogo de filmes:

mysql> ALTER TABLE disco ADD CONSTRAINT chave_p PRIMARY KEY (id);

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE disco ADD CONSTRAINT chave_p PRIMARY KEY (id);

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE tamanho ADD CONSTRAINT chave_p3 PRIMARY KEY (id);

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE disco ADD CONSTRAINT fk_m FOREIGN KEY (cod_marca) REFERENCES marca(id);

Saída gerada:

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE disco ADD CONSTRAINT fk_t FOREIGN KEY (cod_tamanho) REFERENCES tamanho(id);

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

23 Catálogo de filmes:

mysql> ALTER TABLE disco ADD CONSTRAINT fk_e FOREIGN KEY (cod_estojo) REFERENCES estojo(id);

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE filme_disco ADD CONSTRAINT chave_p5 PRIMARY KEY (id);

Saída gerada:

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE video ADD CONSTRAINT chave_p6 PRIMARY KEY (id);

Saída gerada:

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE filme_disco ADD CONSTRAINT fk_v_d FOREIGN KEY (cod_filme) REFERENCES video(id);

Saída gerada:

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE filme_disco ADD CONSTRAINT fk_d_v FOREIGN KEY (cod_disco) REFERENCES disco(id);

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

23 Catálogo de filmes:

mysql> ALTER TABLE genero ADD CONSTRAINT chave_p6 PRIMARY KEY (id);

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE produtora ADD CONSTRAINT chave_p7 PRIMARY KEY (id);

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> CREATE TABLE artista (id int primary key, nome text) ENGINE = INNODB;

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
```

mysql> CREATE TABLE diretor (id int primary key, nome text) ENGINE = INNODB;

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

23 Catálogo de filmes:

mysql> CREATE TABLE filme_artista (id int primary key, cod_filme int, cod_artista int, CONSTRAINT fk_f_a FOREIGN KEY (cod_filme) REFERENCES video(id), CONSTRAINT fk_a_f FOREIGN KEY (cod_artista) REFERENCES artista(id)) ENGINE = INNODB;

Saída gerada:

```
Query OK, 0 rows affected (0.02 sec)
```

mysql> CREATE TABLE filme_diretor (id int primary key, cod_filme int, cod_diretor int, CONSTRAINT fk_f_d FOREIGN KEY (cod_filme) REFERENCES video(id), CONSTRAINT fk_d_f FOREIGN KEY (cod_diretor) REFERENCES diretor(id)) ENGINE = INNODB;

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
```

23 Catálogo de filmes:

Agora finalmente podemos inserir os filmes no nosso banco de dados. Se nossa aplicação utiliza-se colunas NOT NULL para as chaves estrangeiras, seria necessário povoar as tabelas auxiliares (como estojo, tamanho, artista, diretor, etc), para que as principais tabelas (disco e video) pudessem ser povoadas. Embora nossa aplicação não exija isto, vamos povoar algumas tabelas auxiliares, para que nosso banco tenha informações importantes. Deixaremos de lado a parte de artistas, diretores e produtoras. Vamos preencher no momento, apenas as tabelas de informações técnicas. A primeira será a tabela de tamanho de disco, depois alguns modelos de discos e finalmente três estojos:

mysql> INSERT INTO tamanho (descricao) VALUES ('650 MB'), ('700 MB'), ('4.7 GB'), ('9.0 GB');

Saída gerada:

```
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

mysql> INSERT INTO marca (nome) VALUES ('NIPPONIC'), ('ELL COM'), ('PLASMON'), ('migra'), ('ELGIN'), ('PHILIPS'), ('imation'),('DR. Hank'), ('Power Disc');

Saída gerada:

```
Query OK, 9 rows affected (0.01 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

mysql> INSERT INTO estojo VALUES (1), (2), (3);

Saída gerada:

```
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

23 Catálogo de filmes:

Agora vamos povoar a tabela de filmes (lembrando que não setamos como NOT NULL as colunas que são chaves estrangeiras em nenhuma tabela, então podemos deixar todas as chaves em branco, só não podemos preencher as chaves estrangeiras com valores não encontrados em suas tabelas pais). Após a inserção, os filmes não estarão relacionados a nenhum disco. Então inserimos os discos (tabela disco, obviamente) e após, faremos a ligação entre as duas tabelas (video e disco) na tabela filme_disco:

mysql> INSERT INTO video (nome) VALUES ('Casseta & Planeta - Seus Problemas Acabaram'),('Casseta & Planeta - A Taça do Mundo é Nossa'), ('Premonição'), ('Soldado Universal'), ('Meu Nome é Ninguém'), ('O Mestre das Ilusões');

Saída gerada:

```
Query OK, 6 rows affected (0.02 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

Para inserirmos um disco, poderíamos colocar apenas sua id, mas queremos já informar a marca e o tamanho do disco (por padrão setamos o campo, bom_estado como false, mas se o disco estiver em bom estado, alteramos o campo para true). Os filmes inseridos até o momento, são de uma única mídia, um disco da ELL COM, com 4.7 Gigabytes. Para inserirmos o disco que contém estes filmes, podemos setar o tamanho do disco (cod_tamanho) para 3, relativo a 4.7 GB (basta fazer uma seleção na tabela de tamanho). Queremos informar também o nome do disco (ELL COM = 2) na coluna cod_marca. Então, nossa inserção poderia ser assim:

INSERT INTO disco (cod_marca, cod_tamanho, bom_estado) VALUES (2, 3, TRUE);

23 Catálogo de filmes:

Esta inserção não está errada, mas é inconveniente estar acessando outras tabelas para poder fazer a inserção de apenas um disco. Tivemos que recorrer a tabela marca, para saber que o produto ELL COM tem id igual a 2. Tivemos que recorrer a tabela tamanho, para obter o código de um disco de 4.7 GB.
Neste momento se faz interessante o uso de subseleções. Vamos analisar a parte final da inserção (VALUES (2, 3, TRUE)).
Podemos fazer uma seleção que retorne 2 para o cod_marca: SELECT id FROM marca WHERE nome = 'ELL COM'.
Podemos fazer uma seleção que retorne 3 para o cod_tamanho: SELECT id FROM tamanho WHERE descricao = '4.7 GB'.
Agora podemos substituir estes dois números, pelas suas seleções:

mysql> INSERT INTO disco (cod_marca, cod_tamanho, bom_estado) VALUES ((SELECT id FROM marca WHERE nome = 'ELL COM'), (SELECT id FROM tamanho WHERE descricao = '4.7 GB'), TRUE);

Saída gerada:

```
Query OK, 1 row affected (0.03 sec)
```

Os filmes todos fazem parte do mesmo disco, este disco ficou automaticamente com id igual a 1. Agora temos que ligar os filmes ao disco. Essa ligação será simples: Cada filme gravado até o momento, será inserido na tabela intermediária filme_disco e será ligado a um disco. Se verificarmos a tabela filme_disco, ela tem além de sua chave primária, duas chaves estrangeiras (uma para o filme e outra para o disco), vamos alterá-las para NOT NULL. No caso da tabela filme e tabela disco, as chaves estrangeiras poderiam ser nulas, pois talvés não seja interessante preencher todas as colunas para todas as inserções. No caso da tabela intermediária filme_disco, é necessário que todos os campos sejam preenchidos, pois cada linha na tabela deve fazer ligação a um disco e a um filme:

23 Catálogo de filmes:

mysql> ALTER TABLE filme_disco MODIFY cod_filme int NOT NULL;

Saída gerada:

```
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

mysql> ALTER TABLE filme_disco MODIFY cod_disco int NOT NULL;

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Agora vamos fazer o cruzamento dos filmes com o disco:

mysql> INSERT INTO filme_disco (cod_filme, cod_disco) VALUES (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1);

Saída gerada:

```
Query OK, 6 rows affected (0.10 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

Agora temos os filmes cadastrados e ligados ao disco correto (disco 1 no caso). Vamos cadastrar outros filmes em outro disco:

mysql> INSERT INTO video (nome) VALUES ('Click'),('Paixão por Ocasião'), ('O Pássaro Azul');

Saída gerada:

```
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

23 Catálogo de filmes:

Agora vamos fazer a ligação destes filmes ao seu disco:

mysql> INSERT INTO disco (cod_marca, cod_tamanho, bom_estado) VALUES ((SELECT id FROM marca WHERE nome = 'ELL COM'), (SELECT id FROM tamanho WHERE descricao = '4.7 GB'), TRUE);

Saída gerada:

```
Query OK, 1 row affected (0.01 sec)
```

mysql> INSERT INTO filme_disco (cod_filme, cod_disco) VALUES (7, 2), (8, 2), (9, 2);

Saída gerada:

```
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Como podemos ver, nosso segundo disco também é um ELL COM com 4.7 Gigabytes. Usamos no cruzamento dos discos com os filmes (na tabela filme_disco) apenas as chaves, vamos trocar agora pelos nomes dos filmes. O terceiro disco é da marca NIPPONIC:

mysql> INSERT INTO disco (cod_marca, cod_tamanho) VALUES ((SELECT id FROM marca WHERE nome = 'NIPPONIC'), (SELECT id FROM tamanho WHERE descricao = '4.7 GB'));

Saída gerada:

```
Query OK, 1 row affected (0.02 sec)
```

23 Catálogo de filmes:

mysql> INSERT INTO video (nome) VALUES ('O Segredo dos Animais'), ('O Exorcismo de Emily Rose'), ('Inimigo do Estado'), ('O Senhor das Armas');

Saída gerada:

```
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

mysql> INSERT INTO filme_disco (cod_filme, cod_disco) VALUES ((SELECT id FROM video WHERE nome = 'O Segredo dos Animais'), 3), ((SELECT id FROM video WHERE nome = 'O Exorcismo de Emily Rose'), 3), ((SELECT id FROM video WHERE nome = 'Inimigo do Estado'), 3), ((SELECT id FROM video WHERE nome = 'O Senhor das Armas'), 3);

Saída gerada:

```
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

No cadastro de do disco não informamos o estado do DVD. Após verificarmos que o disco está em perfeitas condições, podemos alterar o seu estado.

mysql> UPDATE disco SET bom_estado = TRUE WHERE id = 3;

Saída gerada:

```
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

23 Catálogo de filmes:

Nossos filmes estão todos gravados dentro dos três discos. Os discos pertencem ao primeiro estojo. Vamos ligar os três discos ao primeiro estojo:

mysql> UPDATE disco SET disco.cod_estojo = 1;

Saída gerada:

```
Query OK, 3 rows affected (0.01 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

Esta atualização satisfaz nossa necessidade de alterar os códigos dos discos. Mas é perigoso o seu uso, uma alternativa mais sensata e menos perigosa seria a seguinte instrução:

mysql> UPDATE disco SET disco.cod_estojo = 1 WHERE disco.id >= 1 AND disco.id <= 3;

Saída gerada:

```
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

23 Catálogo de filmes:

Vamos ver agora uma seleção que mostre os nomes dos filmes, a marca, o número e o tamanho do disco:

SELECT * FROM disco, filme_disco, video WHERE filme_disco.cod_filme = video.id AND filme_disco.cod_disco = disco.id;

SELECT * FROM disco, filme_disco, video, tamanho WHERE filme_disco.cod_filme = video.id AND filme_disco.cod_disco = disco.id AND disco.cod_tamanho = tamanho.id;

SELECT * FROM disco, filme_disco, video, tamanho, marca WHERE filme_disco.cod_filme = video.id AND filme_disco.cod_disco = disco.id AND disco.cod_tamanho = tamanho.id AND disco.cod_marca = marca.id;

SELECT video.nome, marca.nome, tamanho.descricao FROM disco, filme_disco, video, tamanho, marca WHERE filme_disco.cod_filme = video.id AND filme_disco.cod_disco = disco.id AND disco.cod_tamanho = tamanho.id AND disco.cod_marca = marca.id;

SELECT video.nome, marca.nome, disco.id, tamanho.descricao FROM disco, filme_disco, video, tamanho, marca WHERE filme_disco.cod_filme = video.id AND filme_disco.cod_disco = disco.id AND disco.cod_tamanho = tamanho.id AND disco.cod_marca = marca.id;

23 Catálogo de filmes:

mysql> SELECT video.nome AS Filme, marca.nome AS Marca, disco.id AS "Número", tamanho.descricao AS Capacidade FROM disco, filme_disco, video, tamanho, marca WHERE filme_disco.cod_filme = video.id AND filme_disco.cod_disco = disco.id AND disco.cod_tamanho = tamanho.id AND disco.cod_marca = marca.id;

Saída gerada:

```
+-----------------------------------------+----------+---------+------------+
| Filme                                   | Marca    | Número  | Capacidade |
+-----------------------------------------+----------+---------+------------+
| Casseta & Planeta - Seus Problemas Acabaram | ELL COM  |       1 | 4.7 GB     |
| Casseta & Planeta - A Taça do Mundo é Nossa | ELL COM  |       1 | 4.7 GB     |
| Premonição                              | ELL COM  |       1 | 4.7 GB     |
| Soldado Universal                       | ELL COM  |       1 | 4.7 GB     |
| Meu Nome é Ninguém                      | ELL COM  |       1 | 4.7 GB     |
| O Mestre das Ilusões                    | ELL COM  |       1 | 4.7 GB     |
| Click                                   | ELL COM  |       2 | 4.7 GB     |
| Paixão por Ocasião                      | ELL COM  |       2 | 4.7 GB     |
| O Pássaro Azul                          | ELL COM  |       2 | 4.7 GB     |
| O Segredo dos Animais                   | NIPPONIC |       3 | 4.7 GB     |
| O Exorcismo de Emily Rose               | NIPPONIC |       3 | 4.7 GB     |
| Inimigo do Estado                       | NIPPONIC |       3 | 4.7 GB     |
| O Senhor das Armas                      | NIPPONIC |       3 | 4.7 GB     |
+-----------------------------------------+----------+---------+------------+
13 rows in set (0.00 sec)
```

Se for necessário trocar a marca do disco (por exemplo, trocar o terceiro disco, para uma mídia ELL COM), podemos fazer a seguinte instrução:

mysql> UPDATE disco SET disco.cod_marca = 2 WHERE id = 3;

Saída gerada:

```
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

24 Stored Procedures:

Basicamente, uma stored procedure é uma série de comandos em linguagem sql para resolver determinado problema. Ficam armazenadas no servidor do MySQL. O recurso de Stored Procedure foi incluído na versão 5 do MySQL. Uma vantagem significativa no uso de Stored Procedures são que elas gastam menos tráfego entre o cliente e o servidor. A parte negativa, fica por conta que a Stored Procedure é executada no servidor, acarretando mais processamento. A estrutura básica de uma Stored Procedure é:

```
CREATE PROCEDURE proc_name([parameters, ...])
[characteristics]
[BEGIN]

    instruções_para_execução;

[END]
```

Para descrevermos as instruções, utilizamos um separador ponto-e-vírgula. Como um ponto-e-vírgula finaliza um comando, devemos alterar o demilitar de comandos, trocando de ponto-e-vírgula para outro caracter ou caracteres (utilizamos aqui barras duplas).
É interessante sobreescrever uma Stored Procedure quando alterarmos alguma coisa no seu corpo.
Vamos mostrar uma função que mostra uma mensagem na tela:

24 Stored Procedures:

```
DELIMITER //
DROP PROCEDURE IF EXISTS ola //
CREATE PROCEDURE ola()
BEGIN
SELECT 'Ola Mundo! Agora em SP no MYSQL!';
END;
//
```

A função seguinte usa parâmetro de entrada.

```
DELIMITER //
DROP PROCEDURE IF EXISTS ola2 //
CREATE PROCEDURE ola2(e_nome varchar(255))
BEGIN SELECT 'Oi Você: ', e_nome ;
END;
//
```

24 Stored Procedures:

Para executar uma Store Procedure basta chama-la através do comando CALL:

```
mysql> DELIMITER //
mysql> DROP PROCEDURE IF EXISTS ola //
```

Saída gerada:

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE PROCEDURE ola()
    -> BEGIN
    -> SELECT 'Ola Mundo! Agora em SP no MYSQL!';
    -> END;
    -> //
```

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

Vamos executar a nossa Store Procedure:

24 Stored Procedures:

mysql> CALL ola//

Saída gerada:

```
+---------------------------------+
| Ola Mundo! Agora em SP no MYSQL! |
+---------------------------------+
| Ola Mundo! Agora em SP no MYSQL! |
+---------------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Lembrando que para executar agora qualquer comando, o finalizador (delimitador) é uma barra dupla. Vamos para a segunda SP:

```
mysql> DELIMITER //
mysql> DROP PROCEDURE IF EXISTS ola2 //
```

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE PROCEDURE ola2(e_nome varchar(255))
    -> BEGIN SELECT 'Oi Você: ', e_nome ;
    -> END;
    -> //
```

Saída gerada:

```
Query OK, 0 rows affected (0.00 sec)
```

24 Stored Procedures:

Para executar a segunda SP, devemos passar um vetor de caracteres:

mysql> CALL ola2('Julio Macke') //

Saída gerada:

```
+------------+-------------+
| Oi Você:   | e_nome      |
+------------+-------------+
| Oi Você:   | Julio Macke |
+------------+-------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Para retornarmos o delimitador ao padrão, basta digitar no terminal: DELIMITER ;
Para verificarmos os procedimentos, podemos utilizar o comando:

mysql> SHOW PROCEDURE STATUS;

Saída gerada:

```
+----------+------+-----------+----------------+---------------------+---------------------+---------------+---------+
| Db       | Name | Type      | Definer        | Modified            | Created             | Security_type | Comment |
+----------+------+-----------+----------------+---------------------+---------------------+---------------+---------+
| catalogo | ola  | PROCEDURE | root@localhost | 2008-05-11 20:59:30 | 2008-05-11 20:59:30 | DEFINER       |         |
| catalogo | ola2 | PROCEDURE | root@localhost | 2008-05-11 21:04:25 | 2008-05-11 21:04:25 | DEFINER       |         |
+----------+------+-----------+----------------+---------------------+---------------------+---------------+---------+
2 rows in set (0.00 sec)
```

24 Stored Procedures:

Quando utilizar:
-->> Quando várias aplicações clientes são escritas em diferentes linguagens ou funcionam em diferentes plataformas, mas precisam realizar as mesmas operações de banco de dados.
-->> Quando a segurança é prioritária. Bancos, por exemplo, usam stored procedures para todas as operações comuns. Isto fornece um ambiente consistente e seguro, e procedures podem assegurar que cada operação seja registrada de forma apropriada. Neste tipo de condiguração, aplicações e usuários não conseguiriam nenhuma acesso as tabelas do banco de dados diretamente, mas apenas podem executar stored procedures específicas.
Stored procedures exigem a tabela proc na banco de dados mysql.

25 Entendendo o Banco de dados:

Vamos analisar a estrutura do servidor. O servidor do MySQL cria duas bases de dados para seu controle. Nessas duas bases podemos opter diversas informações sobre todos os bancos. As bases são: mysql e information_schema.

INFORMATION_SCHEMA:

O banco de dados information_schema apresenta informações diversas do banco. É um tipo de dicionário de dados (armazena as tabelas dos sistema, as visões, os procedimentos, tipos de caracteres, etc):
Estrutura do banco:

```
Tables_in_information_schema

CHARACTER_SETS
COLLATIONS
COLLATION_CHARACTER_SET_APPLICABILITY
COLUMNS
COLUMN_PRIVILEGES
KEY_COLUMN_USAGE
PROFILING
ROUTINES
SCHEMATA
SCHEMA_PRIVILEGES
STATISTICS
TABLES
TABLE_CONSTRAINTS
TABLE_PRIVILEGES
TRIGGERS
USER_PRIVILEGES
VIEWS
```

25 Entendendo o Banco de dados:

-->> TABELA SCHEMATA: Trás informações sobre as bases de dados cadastradas no servidor (nome, codificação de caracteres, etc).
Estrutura da tabela:

```
  Field                          Type           Null   Key   Default   Extra

  CATALOG_NAME                   varchar(512)   YES           NULL
  SCHEMA_NAME                    varchar(64)    NO
  DEFAULT_CHARACTER_SET_NAME     varchar(64)    NO
  DEFAULT_COLLATION_NAME         varchar(64)    NO
  SQL_PATH                       varchar(512)   YES           NULL
```

O comando SHOW DATABASES que estudados anteriormente, pode ser substituído pelo seguinte comando:

mysql> SELECT information_schema.SCHEMATA.SCHEMA_NAME AS "Databases" FROM information_schema.SCHEMATA;

-->> TABELA CHARACTER_SETS: Mantém arquivadas as informações sobre as codificações de caracteres disponíveis.
Estrutura da tabela:

```
  Field                  Type           Null   Key   Default   Extra

  CHARACTER_SET_NAME     varchar(64)    NO
  DEFAULT_COLLATE_NAME   varchar(64)    NO
  DESCRIPTION            varchar(60)    NO
  MAXLEN                 bigint(3)      NO           0
```

25 Entendendo o Banco de dados:

Vamos ver se existe suporte para UNICODE:

```
mysql> SELECT * FROM information_schema.CHARACTER_SETS WHERE information_schema.CHARACTER_SETS.CHARACTER_SET_NAME LIKE '%utf8%';
```

-->> TABELA COLLATIONS: Fornece informações dos idiomas disponíveis sobre cada tipo de caracteres disponível.
Vamos ver os idiomas disponíveis para UTF-8:

```
mysql> SELECT information_schema.COLLATIONS.COLLATION_NAME FROM information_schema.COLLATIONS WHERE information_schema.COLLATIONS.CHARACTER_SET_NAME = 'utf8';
```

25 Entendendo o Banco de dados:

-->> TABELA COLUMNS: Trás as informações (nome, tipo, tamanho, se é chave ou não, comentários, etc) de todas as colunas de todas as tabelas de todos os bancos.
Estrutura da tabela:

Field	Type	Null	Key	Default	Extra
TABLE_CATALOG	varchar(512)	YES		NULL	
TABLE_SCHEMA	varchar(64)	NO			
TABLE_NAME	varchar(64)	NO			
COLUMN_NAME	varchar(64)	NO			
ORDINAL_POSITION	bigint(21)	NO		0	
COLUMN_DEFAULT	longtext	YES		NULL	
IS_NULLABLE	varchar(3)	NO			
DATA_TYPE	varchar(64)	NO			
CHARACTER_MAXIMUM_LENGTH	bigint(21)	YES		NULL	
CHARACTER_OCTET_LENGTH	bigint(21)	YES		NULL	
NUMERIC_PRECISION	bigint(21)	YES		NULL	
NUMERIC_SCALE	bigint(21)	YES		NULL	
CHARACTER_SET_NAME	varchar(64)	YES		NULL	
COLLATION_NAME	varchar(64)	YES		NULL	
COLUMN_TYPE	longtext	NO			
COLUMN_KEY	varchar(3)	NO			
EXTRA	varchar(20)	NO			
PRIVILEGES	varchar(80)	NO			
COLUMN_COMMENT	varchar(255)	NO			

Vamos ver as informações do banco de dados biblioteca_musical e da tabela música:

mysql> SELECT * FROM COLUMNS WHERE TABLE_SCHEMA = 'biblioteca_musical'AND TABLE_NAME = 'musica';

25 Entendendo o Banco de dados:

A anterior acima mostra a relação de todas as colunas da tabela musica. Deverão aparecer cinco resultados. Para vermos apenas uma tabela, podemos especificá-la desta forma:

mysql> SELECT * FROM COLUMNS WHERE TABLE_SCHEMA = 'biblioteca_musical' AND TABLE_NAME = 'musica' AND COLUMN_NAME = 'id';

-->> TABELA TABLES: Grava as informações relativas às tabelas do banco de dados como um todo. Apresenta várias colunas (campos):

Field	Type	Null	Key	Default	Extra
TABLE_CATALOG	varchar(512)	YES		NULL	
TABLE_SCHEMA	varchar(64)	NO			
TABLE_NAME	varchar(64)	NO			
TABLE_TYPE	varchar(64)	NO			
ENGINE	varchar(64)	YES		NULL	
VERSION	bigint(21)	YES		NULL	
ROW_FORMAT	varchar(10)	YES		NULL	
TABLE_ROWS	bigint(21)	YES		NULL	
AVG_ROW_LENGTH	bigint(21)	YES		NULL	
DATA_LENGTH	bigint(21)	YES		NULL	
MAX_DATA_LENGTH	bigint(21)	YES		NULL	
INDEX_LENGTH	bigint(21)	YES		NULL	
DATA_FREE	bigint(21)	YES		NULL	
AUTO_INCREMENT	bigint(21)	YES		NULL	
CREATE_TIME	datetime	YES		NULL	
UPDATE_TIME	datetime	YES		NULL	
CHECK_TIME	datetime	YES		NULL	
TABLE_COLLATION	varchar(64)	YES		NULL	
CHECKSUM	bigint(21)	YES		NULL	
CREATE_OPTIONS	varchar(255)	YES		NULL	
TABLE_COMMENT	varchar(80)	NO			

25 Entendendo o Banco de dados:

Algumas colunas: TABLE_SCHEMA e TABLE_NAME são respectivamente, o nome do banco de dados e a tabela. TABLE_TYPE indica se é uma tabela do sistema ou de usuário ou ainda se é uma visão. ENGINE trás o tipo de tabela.
A consulta a seguir mostra o nome do esquema de dados, o nome da tabela, o tipo e o mecanismo de armazenamento:

mysql> SELECT CONCAT(TABLE_SCHEMA, ".", TABLE_NAME) AS "Banco e Tabela", TABLE_TYPE AS Tipo, ENGINE AS "Engenharia de armazenamento" FROM TABLES ORDER BY TABLE_SCHEMA;

-->> TABELA VIEWS: Armazena as visões criadas em qualquer base de dados. Sua estrutura é simples:

```
  Field                Type              Null    Key    Default    Extra

  TABLE_CATALOG        varchar(512)      YES             NULL
  TABLE_SCHEMA         varchar(64)       NO
  TABLE_NAME           varchar(64)       NO
  VIEW_DEFINITION      longtext          NO
  CHECK_OPTION         varchar(8)        NO
  IS_UPDATABLE         varchar(3)        NO
  DEFINER              varchar(77)       NO
  SECURITY_TYPE        varchar(7)        NO
```

25 Entendendo o Banco de dados:

Para observarmos todas as visões a consulta é simples:

mysql> SELECT * FROM information_schema.VIEWS;

Algumas colunas: TABLE_NAME é o nome da visão criada. TABLE_SCHEMA é o nome do banco de dados envolvido na consulta. VIEW_DEFINITION é a seleção executada (a definição da consulta). DEFINER indica o usuário que criou a visão.
Esta consulta retorna as visões de todos os bancos de dados (temos duas visões em biblioteca_musical e uma em teste1). É mais interessante especificarmos a nossa consulta:

mysql> SELECT TABLE_NAME, TABLE_SCHEMA FROM information_schema.VIEWS WHERE TABLE_SCHEMA = 'biblioteca_musical';

25 Entendendo o Banco de dados:

MYSQL:

Esta base de dados é usada para controle do MySQL. Sua estrutura básica é:

```
Tables_in_mysql

columns_priv
db
func
help_category
help_keyword
help_relation
help_topic
host
proc
procs_priv
tables_priv
time_zone
time_zone_leap_second
time_zone_name
time_zone_transition
time_zone_transition_type
user
```

As duas principais tabelas são: host e user.

-->> TABELA USER: Mostra os usuários registrados no banco. Ao utilizar o parâmetro -u no console para acessar a base, o servidor consulta esta tabela. A estrutura da tabela é a seguinte:

25 Entendendo o Banco de dados:

```
Field                     Type                              Null    Key    Default    Extra

  Host                      char(60)                          NO      PRI
  User                      char(16)                          NO      PRI
  Password                  char(41)                          NO
  Select_priv               enum('N','Y')                     NO             N
  Insert_priv               enum('N','Y')                     NO             N
  Update_priv               enum('N','Y')                     NO             N
  Delete_priv               enum('N','Y')                     NO             N
  Create_priv               enum('N','Y')                     NO             N
  Drop_priv                 enum('N','Y')                     NO             N
  Reload_priv               enum('N','Y')                     NO             N
  Shutdown_priv             enum('N','Y')                     NO             N
  Process_priv              enum('N','Y')                     NO             N
  File_priv                 enum('N','Y')                     NO             N
  Grant_priv                enum('N','Y')                     NO             N
  References_priv           enum('N','Y')                     NO             N
  Index_priv                enum('N','Y')                     NO             N
  Alter_priv                enum('N','Y')                     NO             N
  Show_db_priv              enum('N','Y')                     NO             N
  Super_priv                enum('N','Y')                     NO             N
  Create_tmp_table_priv     enum('N','Y')                     NO             N
  Lock_tables_priv          enum('N','Y')                     NO             N
  Execute_priv              enum('N','Y')                     NO             N
  Repl_slave_priv           enum('N','Y')                     NO             N
  Repl_client_priv          enum('N','Y')                     NO             N
  Create_view_priv          enum('N','Y')                     NO             N
  Show_view_priv            enum('N','Y')                     NO             N
  Create_routine_priv       enum('N','Y')                     NO             N
```

25 Entendendo o Banco de dados:

```
| Alter_routine_priv      | enum('N','Y')                          | NO |  | N |
| Create_user_priv        | enum('N','Y')                          | NO |  | N |
| ssl_type                | enum('','ANY','X509','SPECIFIED')      | NO |  |  |
| ssl_cipher              | blob                                   | NO |  |  |
| x509_issuer             | blob                                   | NO |  |  |
| x509_subject            | blob                                   | NO |  |  |
| max_questions           | int(11) unsigned                       | NO |  | 0 |
| max_updates             | int(11) unsigned                       | NO |  | 0 |
| max_connections         | int(11) unsigned                       | NO |  | 0 |
| max_user_connections    | int(11) unsigned                       | NO |  | 0 |
```

Vamos ver os usuários e seus permissões de acesso:

mysql> SELECT host, user FROM user;

Na coluna host, um asterísco significa acesso de qualquer local. Um endereço de IP, libera o usuário para acessar deste IP. Pode-se colocar os nomes das máquinas em host também.