



Database
HOGWARTS

Maria Clara Alves Acruchi - **mcaa**
Maria Luísa dos Santos Silva - **mlss**
Rebecca Lima Sousa - **rls7**
Tales Vinícius Alves da Cunha - **tvac**

Minimundo

A Escola de Magia e Bruxaria de Hogwarts deseja fazer um Banco de Dados para controlar os bruxos que a compõem.

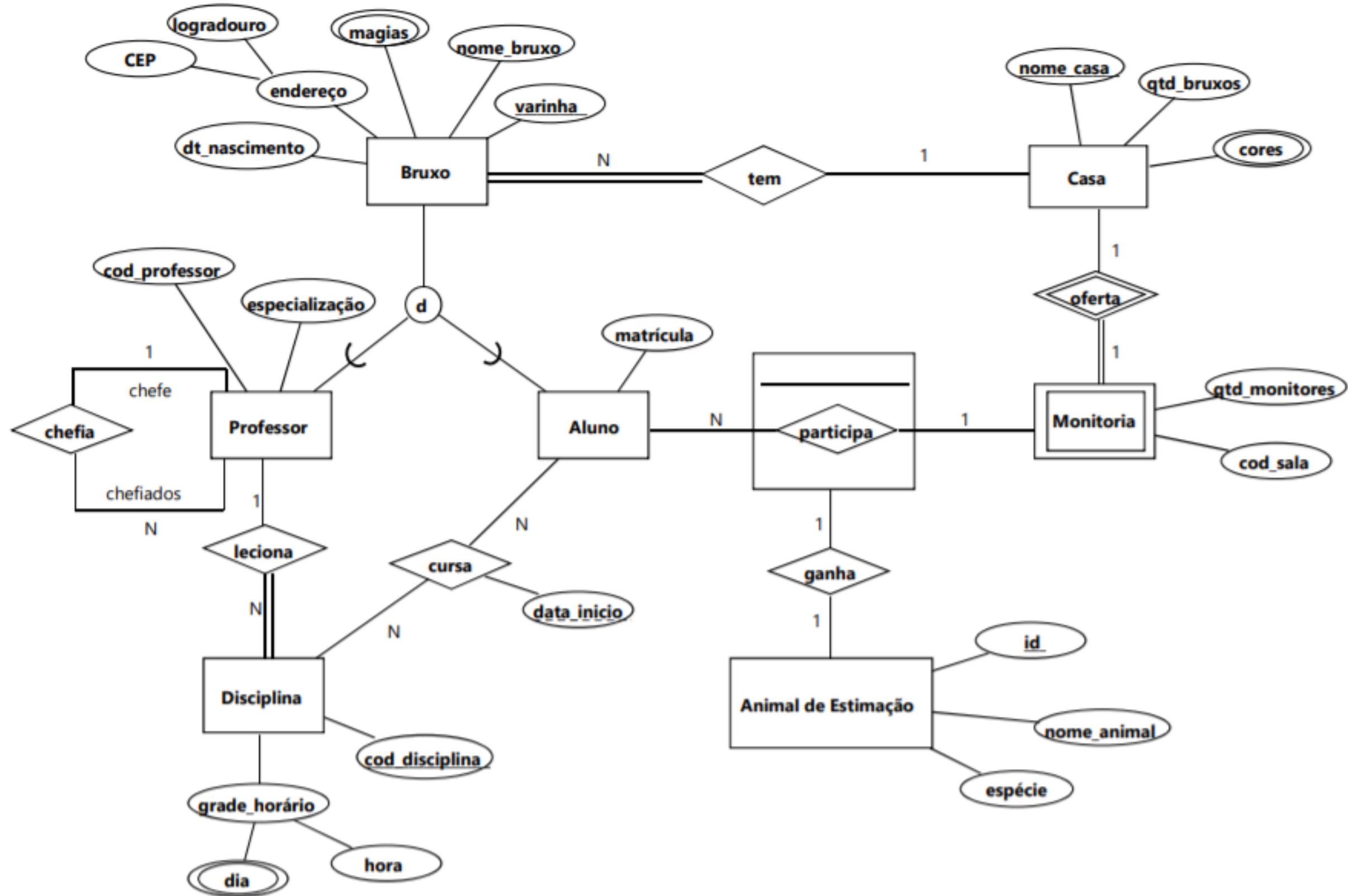
Um **bruxo** (dt_nascimento, varinha, nome_bruxo, magias, endereço (CEP, logradouro)) tem uma **casa** (nome_casa, qtd_bruxos, cores). Além disso, um bruxo pode ser um professor ou um aluno.

Um **professor** (cod_professor, especialização) leciona várias **disciplinas** (cod_disciplina, grade_horario(dia, hora)), as quais são lecionadas por apenas um professor. Além disso, um professor pode ser **chefe** de vários outros professores, os quais só podem ter um único chefe.

Um **aluno** (matrícula) pode **cursar** várias disciplinas, as quais são cursadas por vários alunos. Como um aluno pode cursar várias disciplinas ao longo do tempo, é necessário guardar a data de início em que o aluno cursou determinada disciplina. Um aluno também pode **participar** como monitor de uma única **monitoria** (qtd_monitores, cod_sala), a qual é discriminada por uma determinada casa que só pode ofertar uma única monitoria. Caso o aluno participe da monitoria, pode optar por ganhar um **animal de estimação** (id, nome, espécie).



Projeto Conceitual



Projeto Lógico

- Casa(nome_casa, qtd_bruxos, cor_1, cor_2)
- Disciplina(cod_disciplina, grade_horario_hora, grade_horario_dia_1, grade_horario_dia_2, grade_horario_dia_3, varinha)
varinha → Professor(varinha)
- Animal_de_Estimacao(id, nome_animal, especie)
- Monitoria(nome_casa, qtd_monitores, cod_sala)
nome_casa → Casa(nome_casa)
- Bruxo(varinha, nome_bruxo, dt_nascimento, end_CEP, end_logradouro, nome_casa!)
nome_casa → Casa(nome_casa)
- Magias(varinha, magia)
varinha → Bruxo(varinha)
- Professor(varinha, cod_professor, especializacao, chefe)
varinha → Bruxo(varinha)
chefe → Professor(varinha)
- Aluno(varinha, matricula)
varinha → Bruxo(varinha)
- Participa(varinha, nome_casa!, [id])
varinha → Aluno(varinha)
- nome_casa → Monitoria(nome_casa)
- id → Animal_de_Estimacao(id)
- Cursa(varinha, cod_disciplina, dt_inicio)
varinha → Aluno(varinha)
cod_disciplina → Disciplina(cod_disciplina)

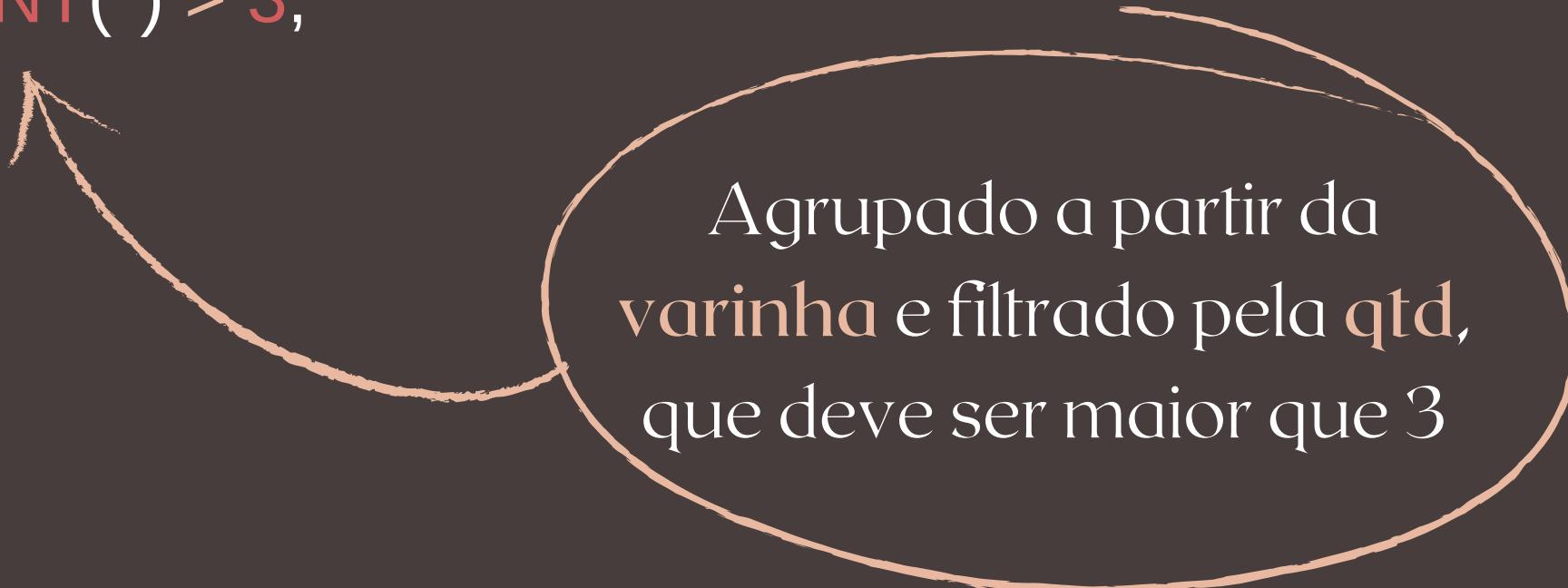
Projeto Físico



Group by/Having

Projetar as varinhas dos alunos que cursaram mais de 3 disciplinas em 1993

```
SELECT varinha, COUNT(*) AS qtd  
FROM cursa  
WHERE EXTRACT(YEAR FROM dt_inicio) = 1993  
GROUP BY varinha  
HAVING COUNT(*) > 3;
```

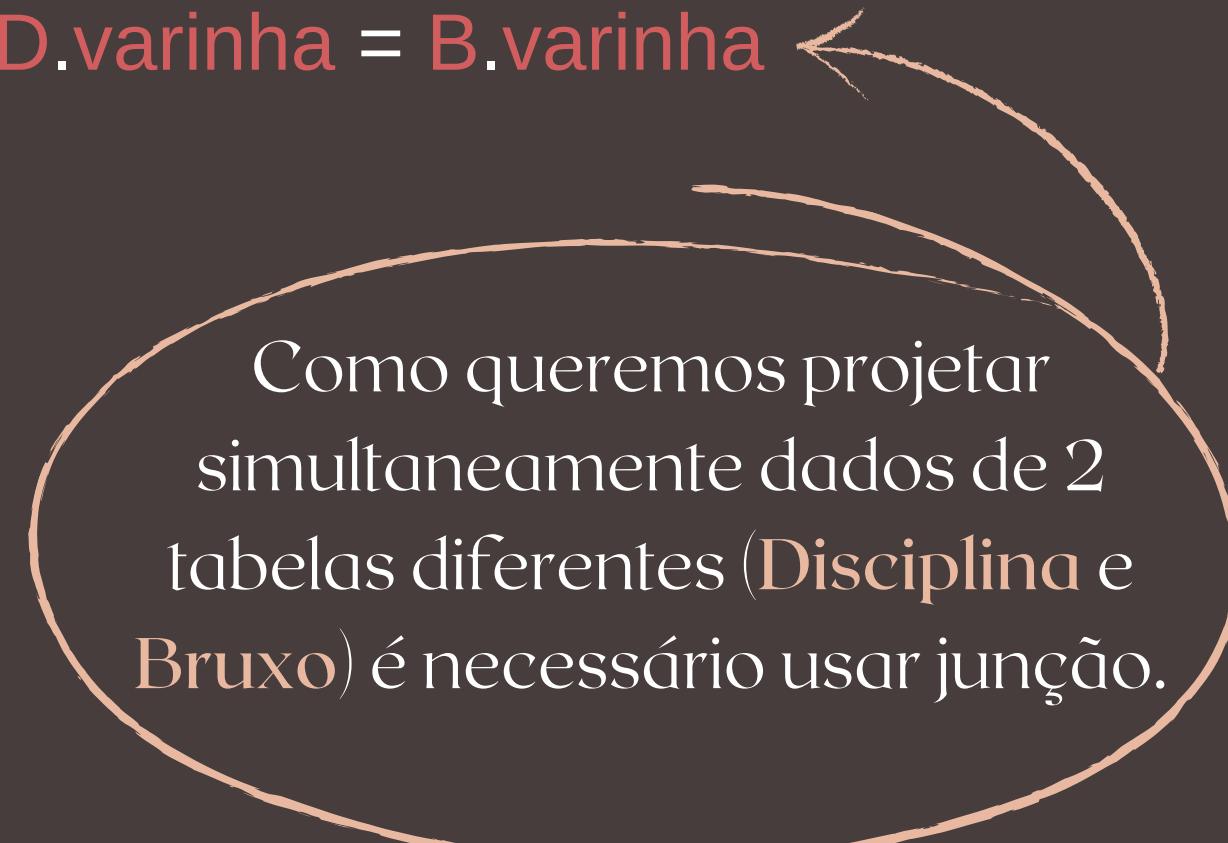


Agrupado a partir da
varinha e filtrado pela qtd,
que deve ser maior que 3

Junção Interna

Projetar os nomes dos professores, os códigos de suas disciplinas e a hora com aulas após o meio dia

```
SELECT B.nome_bruxo, D.cod_disciplina, D.grade_horario_hora  
FROM bruxo B INNER JOIN disciplina D ON D.varinha = B.varinha  
WHERE D.grade_horario_hora > '12:00';
```

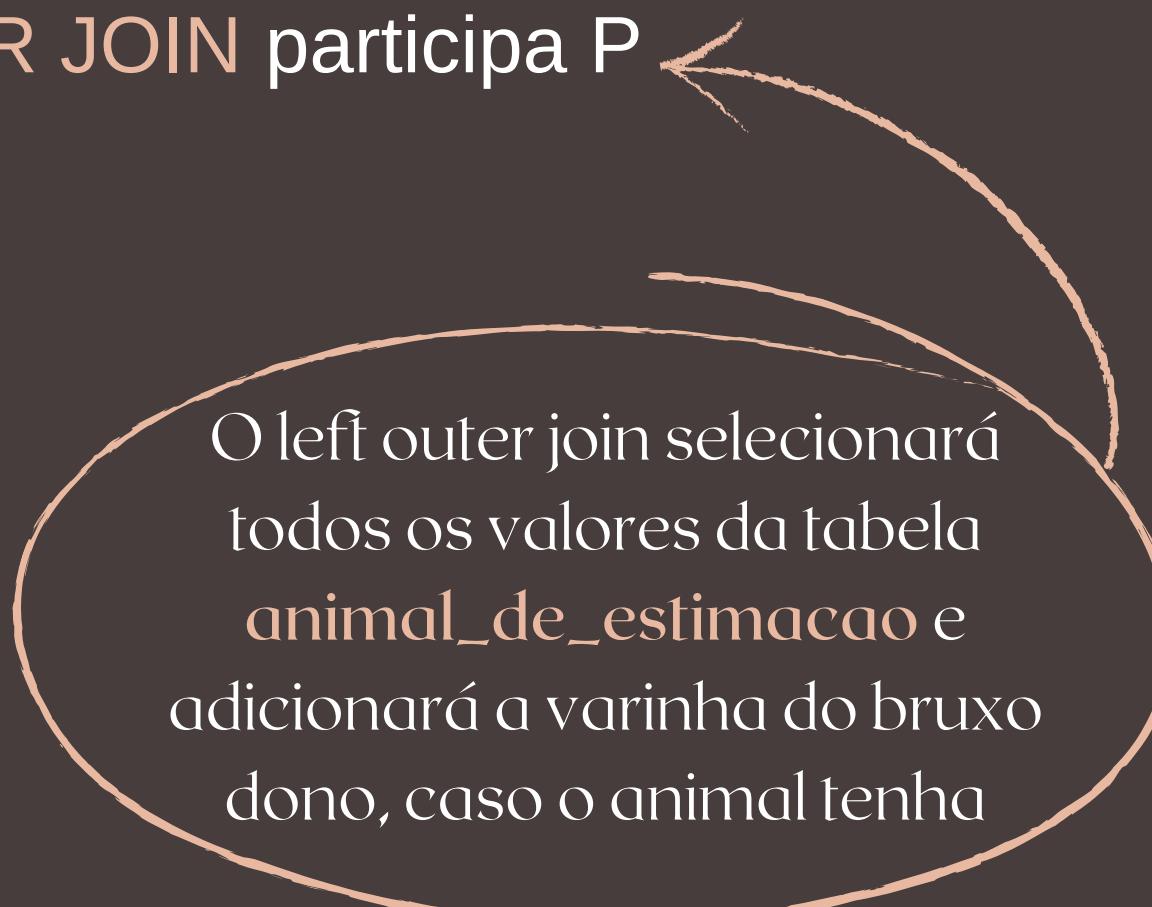


Como queremos projetar simultaneamente dados de 2 tabelas diferentes (**Disciplina** e **Bruxo**) é necessário usar junção.

Junção Externa

Projetar o nome de todos os animais de estimação e a varinha de seus donos (bruxos)

```
SELECT A.nome_animal, P.varinha  
FROM animal_de_estimacao A LEFT OUTER JOIN participa P  
ON P.id_animal = A.id_animal;
```

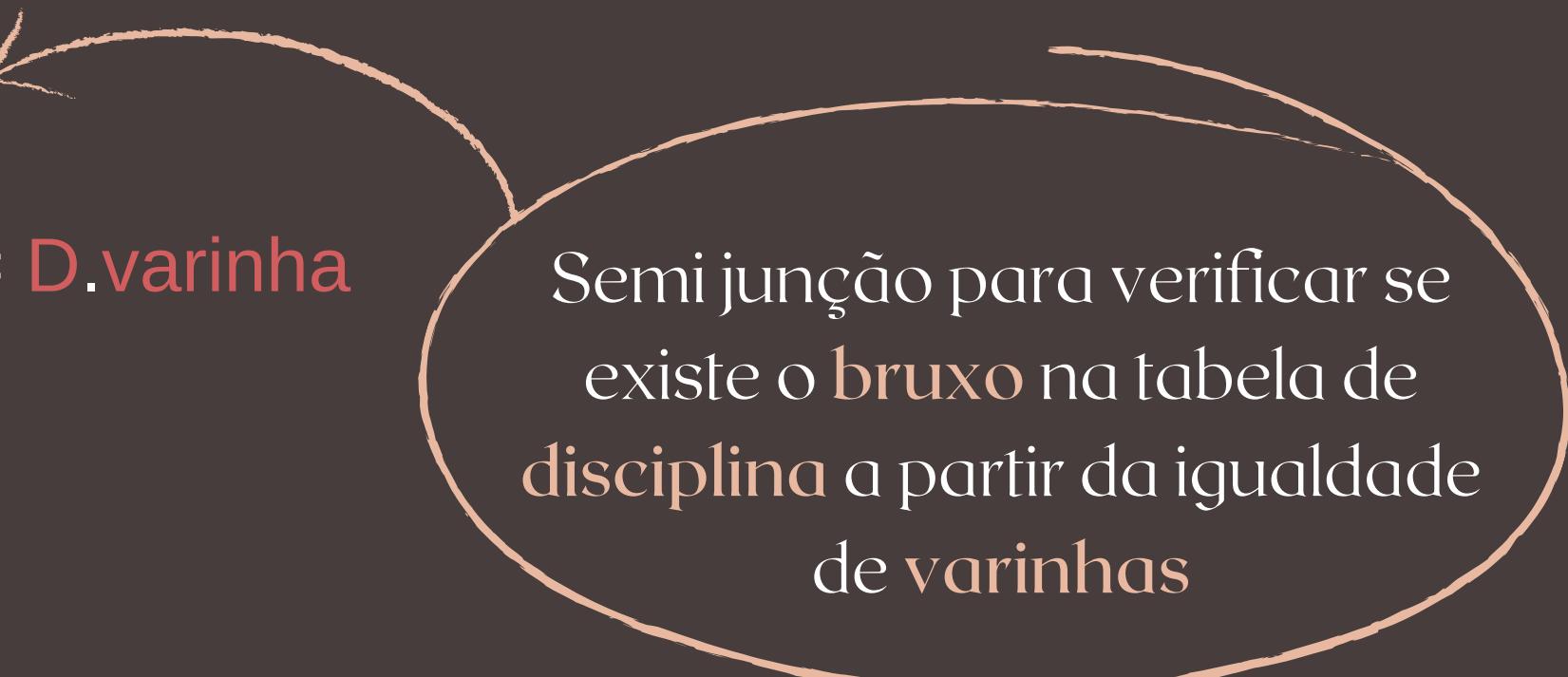


O left outer join selecionará todos os valores da tabela **animal_de_estimacao** e adicionará a varinha do bruxo dono, caso o animal tenha

Semi Join

Projetar o nome dos professores que lecionam alguma disciplina

```
SELECT B.nome_bruxo  
FROM bruxo B  
WHERE EXISTS (  
    SELECT *  
    FROM disciplina D  
    WHERE B.varinha = D.varinha  
);
```



Semi junção para verificar se existe o bruxo na tabela de disciplina a partir da igualdade de varinhas

Anti Join

Projetar a matrícula e o nome dos alunos que não têm animal de estimação

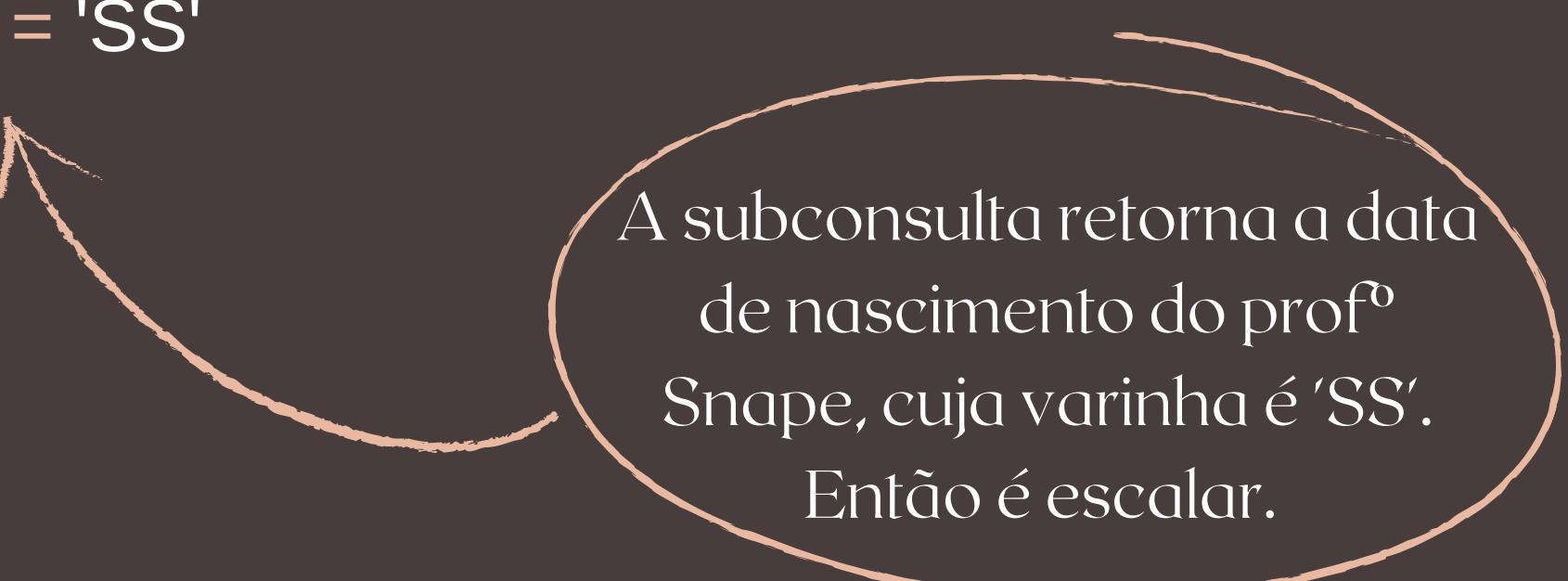
```
SELECT A.matricula, B.nome_bruxo  
FROM aluno A INNER JOIN bruxo B ON B.varinha = A.varinha  
WHERE NOT EXISTS (  
    SELECT *  
    FROM participa P  
    WHERE A.varinha = P.varinha AND P.id_animal IS NOT NULL  
);
```

A subconsulta é do tipo linha porque só pode existir no máximo 1 instância de cada varinha em Participa (varinha é PK de Participa).

Subconsulta do tipo escalar

Projetar todos os nomes e dt_nascimento dos bruxos mais velhos que o professor de varinha = 'SS' (Severo Snape)

```
SELECT B.nome_bruxo, B.dt_nascimento  
FROM bruxo B  
WHERE B.dt_nascimento < (  
    SELECT B1.dt_nascimento  
    FROM bruxo B1  
    WHERE B1.varinha = 'SS'  
);
```

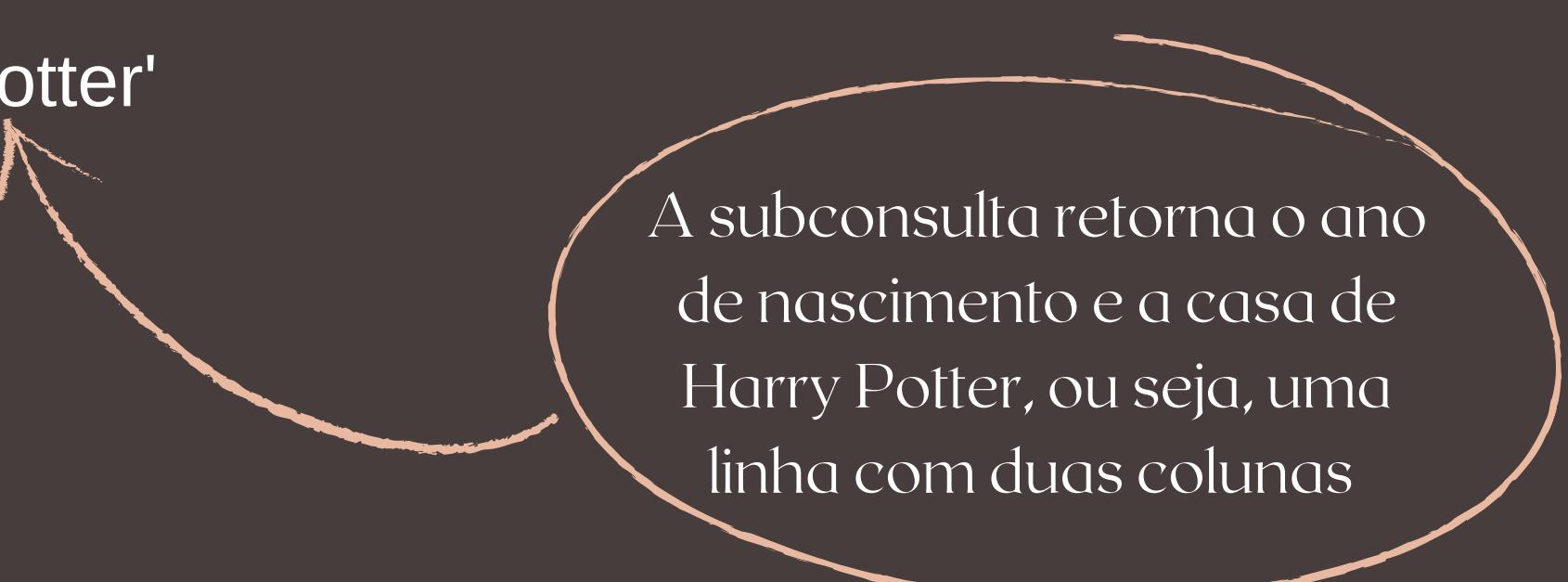


A subconsulta retorna a data de nascimento do profº Snape, cuja varinha é 'SS'.
Então é escalar.

Subconsulta do tipo linha

Projetar bruxos que nasceram no mesmo ano e que são da mesma casa que "Harry Potter" exceto ele mesmo

```
SELECT B.nome_bruxo  
FROM bruxo B  
WHERE B.nome_bruxo <> 'Harry Potter'  
AND (EXTRACT(YEAR FROM B.dt_nascimento), B.nome_casa) = (  
    SELECT EXTRACT(YEAR FROM B1.dt_nascimento), B1.nome_casa  
    FROM bruxo B1  
    WHERE B1.nome_bruxo = 'Harry Potter'  
);
```



A subconsulta retorna o ano de nascimento e a casa de Harry Potter, ou seja, uma linha com duas colunas

Operação de Conjunto

Projetar os códigos dos professores e as matrículas alunos cujas varinhas (iniciais) começam com a letra 'R'

```
SELECT A.varinha AS iniciais, A.matricula AS identifier  
FROM aluno A  
WHERE A.varinha LIKE 'R%'  
UNION  
SELECT P.varinha, P.cod_professor  
FROM professor P  
WHERE P.varinha LIKE 'R%';
```

Procedimento

Procedimento que imprime a varinha dos alunos que cursam uma determinada disciplina

```
CREATE OR REPLACE PROCEDURE cursando (codDisciplina cursa.cod_disciplina%TYPE) IS
    CURSOR cur_alunos IS
        SELECT varinha, dt_inicio
        FROM cursa
        WHERE cod_disciplina = codDisciplina;
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Varinha e data de inicio doa alunos que cursam'||codDisciplina||':');
        FOR reg_cursor IN cur_alunos LOOP
            DBMS_OUTPUT.PUT_LINE('Varinha: '|| reg_cursor.varinha || ', data de inicio: ' || reg_cursor.dt_inicio);
        END LOOP;
    END;
```

```
EXEC cursando('HB');
```

Função

Função para contar a quantidade de magias aprendidas por um determinado bruxo

```
CREATE OR REPLACE FUNCTION qtd_magias (ident VARCHAR) RETURN NUMBER IS
    qtd NUMBER;
BEGIN
    SELECT COUNT(*) INTO qtd
    FROM magias
    WHERE varinha = ident;
    RETURN qtd;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN NULL;
END;
```

```
DECLARE
    I VARCHAR(3);
    O NUMBER;
BEGIN
    I:='RLS';
    O:= qtd_magias(I);
    DBMS_OUTPUT.PUT_LINE('Quantidade de magias: ' || O);
END;
```

Gatilho

Toda vez que adicionar ou deletar um bruxo, atualizar qtd_bruxos da casa

```
CREATE OR REPLACE TRIGGER atualizar_qtd_bruxos
AFTER INSERT OR DELETE ON bruxo
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        -- AUMENTA 1
        UPDATE casa
        SET qtd_bruxos = qtd_bruxos + 1
        WHERE nome_casa = :NEW.nome_casa;
    ELSIF DELETING THEN
        -- DIMINUI 1
        UPDATE casa
        SET qtd_bruxos = qtd_bruxos - 1
        WHERE nome_casa = :NEW.nome_casa;
    END IF;
END;
```

Gatilho

Toda vez que adicionar ou deletar um monitor, atualizar qtd_monitores em monitoria

```
CREATE OR REPLACE TRIGGER atualizar_qtd_monitores
AFTER INSERT OR DELETE ON participa
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        -- AUMENTA 1
        UPDATE monitoria
        SET qtd_monitores = qtd_monitores + 1
        WHERE nome_casa = :NEW.nome_casa;
    ELSIF DELETING THEN
        -- DIMINUI 1
        UPDATE monitoria
        SET qtd_monitores = qtd_monitores - 1
        WHERE nome_casa = :NEW.nome_casa;
    END IF;
END;
```

Gatilho

Checar se a herança professor e aluno é disjunta -> Se for professor, não pode ser aluno

```
CREATE OR REPLACE TRIGGER check_aluno_prof
BEFORE INSERT ON aluno
FOR EACH ROW
DECLARE
   achei NUMBER;
BEGIN
    SELECT COUNT(*) INTOachei
    FROM professor
    WHERE varinha = :NEW.varinha;
    IFachei > 0 THEN
        RAISE_APPLICATION_ERROR(-20205,'ESSE BRUXO NÃO PODE SER UM ALUNO PORQUE JÁ É UM PROFESSOR!!!');
    END IF;
END;
```

Gatilho

Checar se a herança professor e aluno é disjunta -> Se for aluno, não pode ser professor

```
CREATE OR REPLACE TRIGGER check_prof_aluno
BEFORE INSERT ON professor
FOR EACH ROW
DECLARE
    achei NUMBER;
BEGIN
    SELECT COUNT(*) INTOachei
    FROM aluno
    WHERE varinha = :NEW.varinha;
    IFachei > 0 THEN
        RAISE_APPLICATION_ERROR(-20205,'ESSE BRUXO NÃO PODE SER UM PROFESSOR PORQUE JÁ É UM ALUNO!!!');
    END IF;
END;
```