CENTRO PAULA SOUZA

GOVERNO DO ESTADO DE
SÃO PAULO

www.centropaulasouza.sp.gov.br

S Q L

STRUCTURED QUERY LANGUAGE

Programação

- Objetivo
- Histórico
- Características
- Conceitos Iniciais
- A evolução dos SGBDs
- Padronização
- A Álgebra Relacional
- Comandos X Operações

Objetivo

- Capacitar o aluno em:
  - Escrever sentenças SQL usando-as para recuperar dados de Base de Dados Relacionais
  - Usar os comandos INSERT, DELETE e UPDATE para manipular bancos de dados relacionais
  - Construir comandos que processem o uso de funções dos SGBDs de boa presença no mercado
  - Construir comandos de pesquisas encadeados (QUERIES e SUB-QUERIES).

Histórico

- 1970: E.F. Cood – "A Relational Model of Data Large Shared Banks" ➔ Modelo Relacional ⇒ Álgebra Relacional ∴ indica uma Linguagem de Pesquisa;
  - Apresenta a Álgebra Relacional
    um conjunto de símbolos que permite uniformizar o entendimento sobre as operações que podemos realizar sobre as tuplas das tabelas
  - Consequência da Álgebra Relacional:
    - Criação da Linguagem de Manipulação para pesquisa dos dados nos Banco de Dados

- 1974: Donald D. Chamberlin e Raymond Boyce (pesquisadores do "IBM San Jose Research Center"): artigo sugerindo a forma da linguagem de consulta estruturada. A linguagem foi chamada de SEQUEL;

- 1975: Chamberlin, Boyce, King e Hammer apresentam a SQUARE (com símbolos matemáticos), semelhante a SEQUEL (mas esta tinha termos em inglês);

- 1976: Chamberlin apresenta a SEQUEL2 que passa a ser usada como linguagem de consulta para o banco de dados que se pesquisava na IBM, o ***systemaR***: criam-se grupos de usuários, a linguagem começa a evoluir rapidamente ;

Histórico

- 1977: O SystemR-IBM (com SEQUEL/2) torna-se operacional + Oracle (Relacional)
- 1980: Chamberlin sumariza as experiências dos usuários com a linguagem (seu nome a esta altura já era SQL – Structured Query Language).
- 1983: IBM lança o DB2. Outros produtos no mercado: Sybase, Ingres, Progress), SQL é padrão de fato;
- 1986: SQL é homologa como linguagem padrão ANSI para tratamento de dados em BD Relacionais
- 1987: Padrão ANSI é aceito pelo ISO (SQL/86)
- 1989: SQL incorpora características de tratamento de integridade (SQL/89)
- 1992: Os comitês ISO e ANSI apresentam SQL2 (SQL/92)
- 1999 SQL3

Características

- **Amigável** para o usuário, sem ser inconveniente para programadores e analistas de sistemas;
- **Aprendizado fácil** e podendo ser usada em linguagens **procedurais** tais como C, COBOL ou PL/I;
- Pode ser usada como **linguagem de comunicação** entre o usuário e o programador facilitando a comunicação;
- Instrumento amigável para **programadores e analistas** sem perder o rigor matemático da Álgebra Relacional.

Conceitos Iniciais

Tópico	Definição
Banco de Dados	Conjunto de arquivos que armazenam os dados de uma organização
Base de Dados	Conjunto de arquivos de dados e também todos os outros componentes que dão suporte ao funcionamento de um sistema de informação
Tabela	Representação de um dos conjuntos de dados necessários para definir o banco de dados de uma organização. A tabela deve ser correspondente a um conjunto no sentido estrito da matemática. Ou seja, deve conter dados de um só conjunto importante para a realização dos procedimentos e controle das regras de uma organização

Conceitos Iniciais

Tópico	Definição
Tupla	Também denominada LINHA da tabela, é uma coleção de valores que representam uma ocorrência da realidade que está retratada na tabela
Campo	Uma das colunas de uma tabela. De outra forma, é a propriedade associada a um grupo de valores contidos em uma tabela, assim sendo, podemos dizer que, em uma tabela de PRODUTOS, um campo seria a descrição do produto ou outro campo seria a quantidade em estoque
Domínio de valor	Regra que define um intervalo de valores válidos para um campo da tabela.

A Evolução dos SGBDs

- Os SGBD são o resultado do amadurecimento das necessidades empresariais para suporte no desenvolvimento de Sistemas de Informação
- OS SGBD devem dar suporte ao projeto do SI, portanto devem permitir:
  - Criar os arquivos que armazenam os dados e administrar estes arquivos
  - Implementar os programas aplicativos

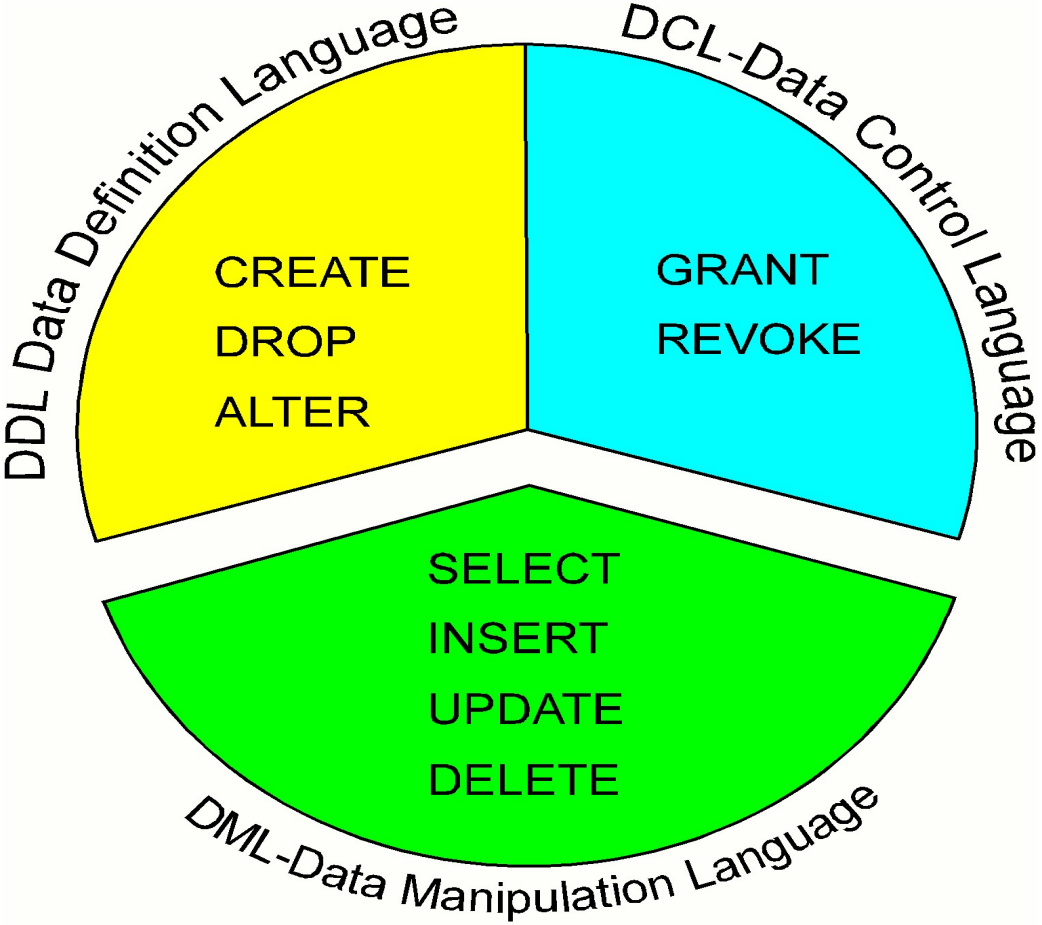
- No grupo de linguagens os SGBDs devem ter seus comandos agrupados em três categorias:

Grupo	Descrição
DDL *Data Defination Language*	Permite que sejam caracterizados a estrutura dos arquivos onde se gravam os dados da organização. É possível definir o tamanho dos campos, do arquivo como um todo e seus índices associados
DCL *Data Control Language*	Linguagem de Definição de Usuários e Grupos de usuários e suas atribuições de acesso aos Dados.
DML *Data Manipulation Language*	Linguagem de Manipulação de Dados. É o grupo de comandos que permite incluir uma nova linha em uma tabela, pesquisar uma linha, alterar alguns dos valores de seus campos ou mesmo remover a linha da tabela

Padronização SQL

- A SQL foi estudada por vários grupos, logo se estabeleceram padrões:
  - ANSI em 1986 e ISO em 1987
  - ANSI em 1992 (denominada SQL:1992)
  - ANSI 1999 e 2003 (ampliação da SQL:1992)
    - Na SQL:1999 já define o padrão para triggers.
    - Na SQL:2003 introduz-se os tipos de dados não-escalonados e algumas características de programação orientada a objetos.

- Os grupos de comandos da SQL são:



Lembrando da Álgebra Relacional

- **Tabela** é um conjunto de **TUPLAS;**
- **Álgebra Relacional** ⇒ grupo de operações que executam sobre as tabelas uma transformação igual àquela apresentada na teoria dos conjuntos;
- As operações da Álgebra Relacional são:
  - UNIÃO — **R ← A ∪ B**
  - INTERSECÇÃO — **R ← A ∩ B**
  - SUBTRAÇÃO — **R ← A – B**
  - PRODUTO CARTESIANO — **R ← A x B**
  - PROJEÇÃO — **R ← A [[C1, C2,..., Cn]]**
  - JUNÇÃO — **R ← A [[Condição]] B**
  - SELEÇÃO — **R ← A [[Condição]]**

Comandos SQL - SELECT

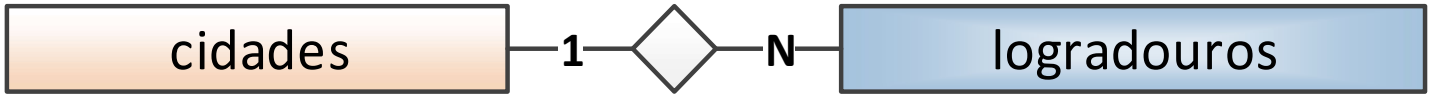
- Implementa as sete operações da Álgebra Relacional
- Tem uma estrutura rígida mas com partes opcionais

```
SELECT [ALL | DISTINCT [ON (expressao[,...])]]
       * | Campo1[AS nome_saída1][,...]
       FROM TAB1[, TAB2, TAB3, ... , TABn]
       [WHERE condição]
       [GROUP BY chave1[,chave2,...]]
           [HAVING condição[,...]] [{} select]
       [ORDER BY campo1 [ASC|DESC|USING operador][,
                 campo2 [ASC|DESC|USING operador][,
                ...campon];
```

- Tudo o que está entre [] é opcional
- O comando SELECT implementa: Seleção, Projeção, Junção (fechada) e Produto Cartesiano.
- Junções Abertas, União, Subtração e Intersecção são implementas usando *operadores* ou *subcomandos*.

Nosso Modelo de Estudo

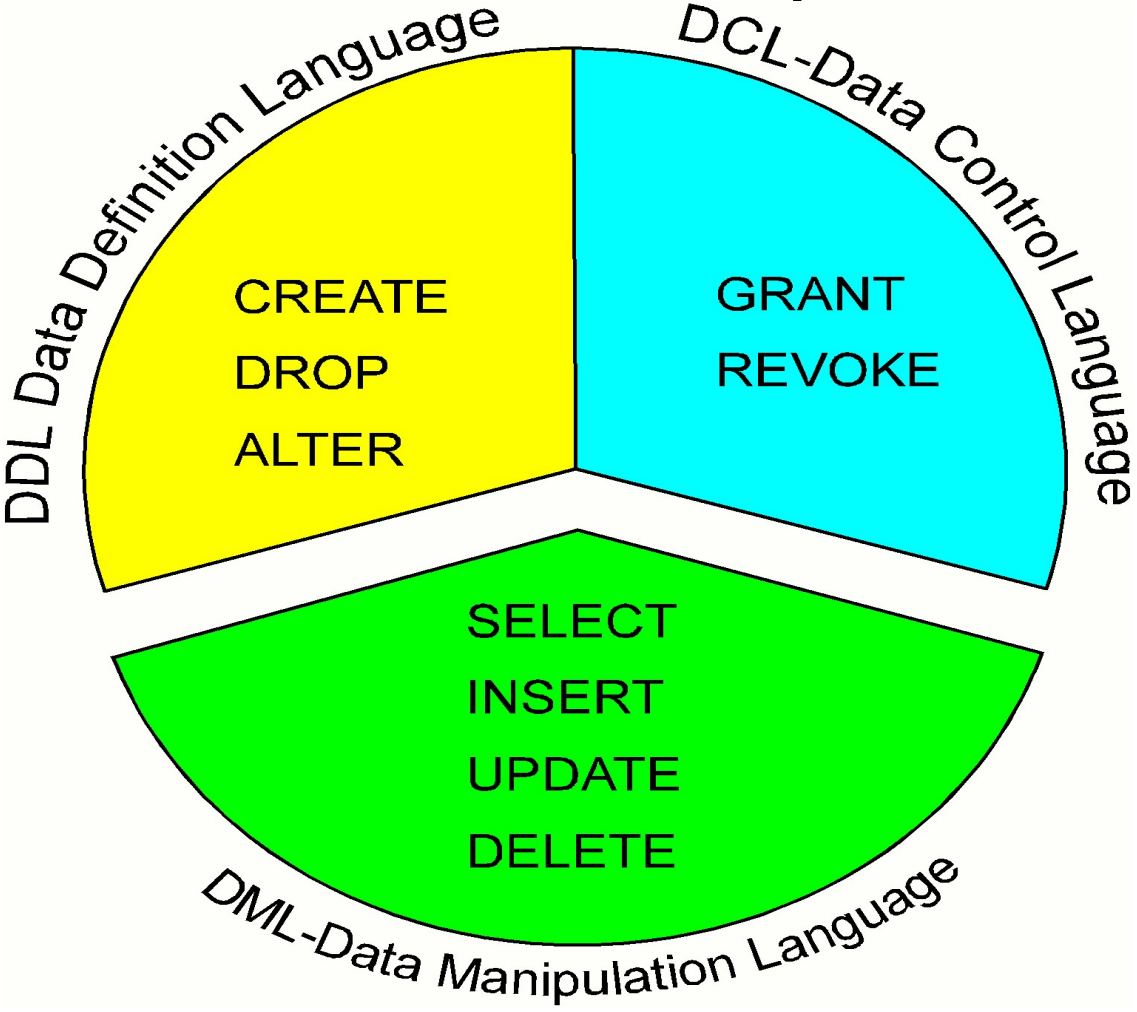
- Usaremos uma base de dados implementada segundo estes modelos:



- Os Comandos que criam estas tabelas (e outras) estão disponibilizados no site da disciplina.
- Crie uma base de dados com nome: LBDPG usando o programa: pgAdmin 4.
- Carregue e execute o Script criando as tabelas.
- Sobre estas tabelas escreva comandos que realizam as seguintes operações:
  - Seleção
  - Projeção
  - Produto Cartesiano
  - Junção Fechada
  - Junção Aberta
  - Junção completa
- Copie os comandos que você escrever em um Arquivo TXT.

Lembrando da Padronização SQL

- Os grupos de comandos da SQL são:



Comandos SQL - SubQueries

- Em SQL podemos combinar o resultado de um comando com outros comandos.
- Isso se chama SUB-Queries (aninhamento).
- Usa-se os parênteses como forma de colocar em um NINHO o resultado de processamentos de comandos;
- Cada comando faz exatamente uma operação. Desta forma a expressão em álgebra relacional:
- **R ← Clientes[[Cidade=”Bauru”]]**
- Deve ser vista como dois comandos em SQL:
  - O primeiro ***CRIA*** a tabela R
  - O segundo ***FAZ*** a seleção e copia as tuplas para R

- Os grupos podem ser definidos assim:

Parte	Ação
SELECT	Indica ao núcleo que deverá ser iniciado o processamento de leitura de dados das tabelas
*| campo [as c1]	Indica quais campos devem ser exibidos no resultado do comando. Se usarmos * vemos todos os campos processados, senão devemos indicar quais queremos ver. O uso do "as" permite o renomear os campos dinamicamente.
FROM	Especifica a(s) tabela(s) a processar. No caso de mais de uma tabela, os nomes devem ser separados por ','
WHERE	Especifica uma condição do processamento das linhas da tabela. Pode-se combinar condições com operadores lógicos. Usa-se operadores de comparação específicos de cada SGBD. Os operadores Padrão ANSI/99 são: =, <>, >, <, >=, <=, IN, BETWEEN e os operadores lógicos são: AND, OR, NOT, XOR.

- Os grupos podem ser definidos assim:

Parte	Ação
GROUP BY	Permite estabelecer agrupamentos de valores que podem apresentar a execução de funções escalares ou aritméticas
HAVING	Executam funções sobre valores de campos, podem indicar resultados que podem ser usados como parâmetro de agrupamento.
ORDER BY	Classificam os dados das tuplas segundo campos da tabela permitindo estabelecer organizações para execução de funções. As chaves de ordenação podem apresentar sub-chaves de ordenação e, para cada uma delas, ter ordens diferentes indicadas pelo uso de ASC ou DESC seguindo cada chave declarada.

- Pode-se escrever, na base de dados, o comando:
  - **SELECT * FROM funcionarios WHERE cedepto='A01';**

- CRIAÇÃO de Tabelas
  - Sintaxe:

CREATE TABLE NomeDaTabela AS
(Comando SQL);

Também pode ser assim:

CREATE TEMPORARY TABLE
NomeDaTabela AS (Comando SQL);

- Combinando uma SELEÇÃO com CREATE
  - **CREATE TABLE func1 AS**
    (SELECT *
    FROM funcionários
    WHERE cedepto='A01');
  - Neste comando a tabela fica gravada no HD e é registrada no Dicionário de Dados da base acessada pelo usuário.

- Ou também:
  - **CREATE *TEMPORARY* TABLE depto01 AS**
    (SELECT * FROM departamentos
    WHERE cedeptosuperior IN
    ('A01','B01'));

  - Neste comando a tabela será criada em memória (no pedaço de memória que o usuário pode acessar no servidor)

- CRIAÇÃO de Tabelas
  - Podemos, também, montar o esquema de uma tabela indicando cada um de seus campos
  - Sintaxe:
  -
```
CREATE TABLE
       NomeTab
       (CAMPO1 TIPO(TAM) CaractComplem[,
        CAMPO2 TIPO(TAM) CaractComplem,
        …
        CAMPOn TIPO(TAM) CaractComplem[,
                  PRIMARY KEY (CAMPO1)]] )
      [ENGINE=InnoDB]
      [DEFAULT CHARSET=utf8]
      [COLLATE=utf8_bin];
```

Comandos SQL – criando tabela (MySQL)

- Um exemplo...
- Para criar as tabelas usamos o CREATE TABLE:

```
CREATE TABLE
        regioes (IDRegiao int(3) NOT NULL,
                 NMRegiao varchar(15) NOT NULL,
                 QTAreaestimada double(12,3) NOT NULL,
        PRIMARY KEY (IDRegiao))
        ENGINE=InnoDB
        DEFAULT CHARSET=utf8
        COLLATE=utf8_bin;
```

Comandos SQL – criando tabela (MySQL)

- Este é um comando DDL da SQL já escrito para o SGBD MySQL.
- Note a especificação do ENGINE indicando o tipo InnoDB.
- A característica de adotar a tabela de símbolos UTF8-BIN somente informa como o SGBD deve tratar os símbolos da tabela de caracteres da língua nacional e de como isso é gravado nas tabelas e lidos a partir delas.
- Calma o script que cria a base está pronto e vamos distribuir.

Comandos SQL - Projeção

- Sintaxe:

  Select NomeDoCampo FROM TABELA;

- Executando na Base:

```
SELECT cpfuncionario, txprimeironome, nuramal
        FROM funcionarios;
```

- Combinado com o Create:

```
CREATE TABLE ResProdutos AS
    (SELECT cpproduto, txnome, vlpreco
        FROM produtos);
```

Comandos SQL – Produto Cartesiano

- Sintaxe:

```
SELECT  *  FROM  TABELA1, TABELA2;
```

Combinado com o Create (exec. na base):

```
CREATE TABLE possiveisturmas AS
   (SELECT * FROM disciplinas, professores);
```

Comandos SQL – Junção (Fechada ou Natural)

- Sintaxe:
- SELECT * FROM TABELA1, TABELA2
  WHERE
  TABELA1.Campo1=TABELA2.Campo1;
- Ou também:
  SELECT * FROM Tab1 INNER JOIN Tab2
  ON
  TABELA1.Campo1=TABELA2.Campo1;
- Ou também:
  SELECT * FROM Tab1 INNER JOIN Tab2 USING (campo);
- Exemplos:

```
Select * FROM  departamentos, funcionarios
        WHERE
            departamentos.cpdepto=funcionarios.cpdepto;

Select * FROM  departamentos INNER JOIN funcionarios
         WHERE
            departamentos.cpdepto=funcionarios.cpdepto;

Select * FROM  departamentos INNER JOIN funcionarios
         USING (cpdepto);
```

- Testando a Junção...
- Criamos uma Tabela com dept com o comando:

```
CREATE TABLE dept AS
        (SELECT * FROM departamentos
                WHERE cpdepto IN ('A01','B01','D11','E01')
```

- Depois criamos duas novas tabelas na Base com os comandos:
-
```
create table func1 as
        (Select * from funcionarios where cedepto='A01');
```

-
```
create table func2 as
        (Select * from funcionarios where cedepto='D11');
```

  - Note: a tabela func1 tem somente os funcionários do departamento 'A01' e func2 os funcionários do departamento 'D11'.

Comandos SQL – Junção (Fechada ou Natural)

- Podemos também fazer algumas coisas mais complexas: "quais são os nomes de gerentes e seus subalternos para os departamentos com código do gerente entre 230, 260 ou 290? "
  - Podemos escrever:

```
SELECT F1.cpFuncionario, F1.txprimeironome,
    D1.cpdepto, D1.txnomedepto, D1.cpFuncionarioGerente,
    F2.cpFuncionario, F2.txprimeironome
    FROM funcionarios as F1, departamentos AS D1, funcionarios AS F2
    WHERE F1.cpfuncionario=D1.cpfuncionario_gerente AND
        D1.cpdepto=F2.cpdepto AND
        F1.cpfuncionario IN ('230','260','290');
```

Comandos SQL – Junção Aberta pela Direita

- Sintaxe:

```
Select * FROM TABELA1 RIGHT JOIN TABELA2
            ON TABELA1.Campo1=TABELA2.Campo1
```

Testando...

- **Podemos saber quais os dept que não tem ligação com func1 através do comando:**

- **SELECT IDFuncionario,TXPrimNome,TXIniciaisMedias,**
  TXSobreNome,func1.IDDepto,dept.IDDepto,TXNomedepto
  FROM func1 RIGHT JOIN dept ON func1.iddepto=dept.iddepto;

- Sintaxe:

```
Select * FROM TABELA1 LEFT JOIN TABELA2
          ON TABELA1.Campo1=TABELA2.Campo1
```

Testando...

- **Podemos saber quais os func1 que não são gerentes de dept com o seguinte comando:**

-
```
SELECT F1.IDFuncionario, F1.TXPrimNome, F1.TXSobreNome,
     D1.iddepto, D1.TXNomedepto, D1.IDFuncGerente
     FROM func1 as F1 LEFT JOIN departamentos AS D1
          ON F1.idfuncionario=D1.idfuncgerente
     WHERE D1.iddepto IS NULL;
```

Comandos SQL – Junção Completa (aberta pelos dois lados)

- Sintaxe:

```
Select * FROM TABELA1 FULL OUTER JOIN TABELA2
          ON TABELA1.Campo1=TABELA2.Campo1
```

Testando...

- **Podemos saber quais os func1 que não são gerentes de dept com o seguinte comando:**

- ```
  SELECT F1.IDFuncionario, F1.TXPrimNome, F1.TXSobreNome,
       D1.iddepto, D1.TXNomedepto, D1.IDFuncGerente
       FROM func1 as F1 LEFT JOIN departamentos AS D1
            ON F1.idfuncionario=D1.idfuncgerente
       WHERE D1.iddepto IS NULL;
  ```
  ```

Comandos SQL - União

- Sintaxe:
  - SELECT * FROM tabela1 UNION (SELECT * FROM tabela2);
- Executando na Base:
  - create table func1 as
    (Select * from funcionarios where cedepto='A01');
  - create table func2 as
    (Select * from funcionarios where cedepto='D11');
  - SELECT * FROM func1 UNION (SELECT * FROM func2);

Comandos SQL - União

- O comando UNION já trata a NÃO repetição de tuplas na tabela resultado.
- Se quisermos recuperar as linhas repetidas devemos escrever UNION ALL na operação, desta forma:

```
Select * FROM  TABELA1
    UNION ALL
   (Select * FROM TABELA2);
```

Testando...

```
Select * FROM  func1 UNION ALL (Select * FROM func2);
```

Comandos SQL - Subtração

- Sintaxe:

```
Select * FROM  TABELA1
     MINUS | EXCEPT
          (Select FROM TABELA2);
```

Combinado com o Create:

```
CREATE TABLE FUNC3 as
   (SELECT * FROM funcionarios MINUS
    (SELECT * FROM func1));
```

Comandos SQL - Subtração

- O MySQL não implementa o operador MINUS, podemos fazer a subtração assim:

```
SELECT * FROM  TAB1
    WHERE CONCAT(todos os campos de TAB1) NOT IN
        (SELECT CONCAT(todos os campos de TAB) FROM TAB2);
```

- Ou podemos usar:

```
SELECT func1.* FROM func1 LEFT JOIN func2 USING
(cpfuncionario) WHERE func2.cpfuncionario IS NULL;
```

- Ou ainda usando o operador IN (ou NOT IN)

```
SELECT * FROM funcionarios
    WHERE idfuncionario NOT IN (select idfuncionario FROM func1);
```

Note bem: parece que com o NOT IN as tabelas TAB1 e TAB2 NÃO precisam ser UNIÃO COMPATÍVEL para fazer o MINUS, basta envolver alguns campos (PK de preferencia) no operador NOT IN.

Agora Interprete este comando:

```
Select * from Funcionarios WHERE idfuncionario NOT IN (SELECT T1.idfuncionario
    FROM (SELECT * from func1 UNION (SELECT * FROM FUNC2)) AS T1);
```

Comandos SQL - Intersecção

- Sintaxe:

```
SELECT * FROM  TAB1
   INTERSECT
    (SELECT * FROM  TAB2);
```

Combinado com o Create:

-
```
CREATE TEMPORARY TABLE resultado AS
 (SELECT * FROM  TAB1
   INTERSECT
    (SELECT * FROM  TAB2);
```

Comandos SQL - Intersecção

- Sintaxe:

```
SELECT * FROM TABELA1
    INTERSECT
        (SELECT * FROM TABELA2);
```

- Nota:
  - Alguns SGBDs seguem o padrão SQL:1992 MAS NÃO implementam este operador
  - Para cada SGBD temos que estudar e ver como implementa a intersecção.

- Nota:
  - No MySQL (5.+) podemos escrever assim:

```
SELECT * FROM func1 WHERE  cpfuncionario
            IN
            (SELECT cpfuncionario FROM func2);
```

  - PORÉM este operador (IN) permite a comparação de uma coluna necessitando que as duas tabelas tenham os campos de comparação MAS não necessariamente sejam UNIÃO COMPATÍVEIS. Isso quebra o conceito de operação intersecção da Álgebra Relacional.
  - É possível se implementar a INTERSECÇÃO desde que se use o IN comparando TODOS os campos das tabelas, usando a função CONCAT().

Comandos SQL - Intersecção

- Uma outra forma de escrever um comando de intersecção é a seguinte:

```
Select * FROM TABELA1
   WHERE EXISTS
     (SELECT * FROM TABELA2  WHERE
         TABELA1.Campo1=TABELA2.Campo1 AND
         TABELA1.Campo2=TABELA2.Campo2 AND
          …… AND
         TABELA1.Campon=TABELA2.Campon);
```

- Note que é mais trabalhoso pois necessita que todos os campos dos esquemas sejam declarados. Além disso alguns SGBDs já não mais implementam este comando.

Comandos SQL - Intersecção

- Na base, podemos escrever:
- **CREATE TABLE func1 AS (Select * from funcionarios WHERE cpdepto IN ('A01','B01');**
- **CREATE TABLE func2 AS (Select * from funcionarios WHERE cpdepto IN ('A01','C01');**
- E a seguir…

Comandos SQL - Intersecção

```
SELECT * FROM func1 WHERE EXISTS
  (SELECT * FROM func2  WHERE
        func1.cpfuncionario = func2.cpfuncionario AND
        func1.tx_primeiro_nome = func2.tx_primeiro_nome AND
        func1.tx_iniciais_medias = func2.tx_iniciais_medias AND
        func1.tx_sobre_nome = func2.tx_sobre_nome AND
        func1.cpdepto = func2.cpdepto AND
        func1.nu_ramal = func2.nu_ramal AND
        func1.dt_contratacao = func2.dt_contratacao AND
        func1.cpfuncao = func2.cpfuncao AND
        func1.cpnivel_educacao = func2.cpnivel_educacao AND
        func1.ao_sexo = func2.ao_sexo AND
        func1.dt_nascimento = func2.dt_nascimento AND
        func1.tx_resenha = func2.tx_resenha AND
        func1.vl_salario = func2.vl_salario AND
        func1.vl_bonus = func2.vl_bonus AND
        func1.vl_comissao = func2.vl_comissao AND
        func1.nu_cep = func2.nu_cep AND
        func1.dt_cadastro = func2.dt_cadastro)
```

- Trabalhando com Funções
  - As funções retornam um único valor cada vez que são chamados. A SQL padrão fornece funções divididas em dois grupos:
    - Escalares
      - Funções que manipulam tipos de dados data e hora, strings, e números, bem como recuperar informações do sistema, como o usuário atual ou o nome de login, etc.
    - Aritméticas
      - Funções para processamentos matemáticos. Por exemplo: ABS, PI, SQUARE, COS, POWER e TAN, entre outras.

- Trabalhando com Funções
  - De modo geral as funções são usadas no comando SELECT, realizando o processamento enquanto "lê" as tuplas de uma tabela.
  - Pode ser usada também com comandos de atualização (UPDATE ou INSERT) informando valor para os campos, ou no comando DELETE (usando para determinar quais tuplas serão deletadas)

- Trabalhando com Funções
  - Exemplificando...
    - Queremos saber quantos funcionarios de cada departamento ganham mais do que 25000 anualmente:
    - ```
      SELECT cpdepto,
             COUNT(cpfuncionario) as "QtdFunc"
             From funcionarios
             WHERE vl_salario>25000
             GROUP BY cpdepto
             ORDER BY cpdepto
      ```
  - Escrevendo outros exemplos
    - Teste alternativas ao comandos apresentados.
      ```

- Trabalhando com Funções
  - Exemplificando...
    - Queremos saber o total médio gasto em vendas para cada clientes. Podemos escrever:
    - ```
      SELECT cpcliente,
             AVG(vl_total_nf_venda)
             FROM nf_vendas
             GROUP BY cpcliente
             ORDER BY cpcliente
      ```
  - Se quisermos o valor do AVG arredondado com duas casas decimais usamos a função ROUND (do MySQL e não no SQL:92)
      ```

- Trabalhando com Funções
  - SELECT cpcliente,
    ROUND(AVG(vl_total_nf_venda),2)
    FROM nf_vendas
    GROUP BY cpcliente
    ORDER BY cpcliente
  - Escrevendo outros exemplos
    - Teste alternativas ao comandos apresentados.

- Trabalhando com Funções
  - E o que será que este comando faz?
    - ```
      SELECT cpcliente AS Cliente,
             SUM(vl_total_nf_venda) AS Total,
             COUNT(cpnf_venda) AS QtdDeNotas,
             AVG(vl_total_nf_venda) AS VlMédio
             FROM nf_vendas
             GROUP BY cpcliente
             ORDER BY cpcliente
      ```
    - Calcula o total de valor em notas de cada Cliente, contas quantas notas foram emitidas e o Valor médio gasto por nota.
    - Que conclusões se pode obter com esta análise?
      ```

Atualização: INSERT

- Sintaxe:

```
INSERT   INTO   <Tab>   [  (Campo1, Campo2, ...)  ]
    VALUES   ('valor docampo1', 'valordocampo 2', ...)
                [,('valordocampo 1, 'valordocampo 2', ...),
                  ('valordocampo 1', 'valordocampo 2', ...),
                  ('valordocampo 1', 'valordocampo 2', ...)];
```

- Exemplo:

```
INSERT INTO DEPARTAMENTOS
     VALUES ( 'A11','CONTROLE DE CUSTOS1','40','A01','-'),
                ( 'A12','CONTROLE DE CUSTOS2','120','A01','-');
```

Atualização: INSERT

- Sintaxe:

```
INSERT   INTO   <Tab>
    SET  Campo1='valordocampo1', Campo2=valordocampo2 , ... ;
```

- Exemplo:

```
INSERT INTO departamentos
    SET
        DEPTNO='A02',
        DEPTNAME='Prova que dá certo',
        MGRNO='000090',
        ADMRDEPT='A01',
        LOCATION='-'   ;
```

Atualização: INSERT com sub-queries

- O Comando de **INSERÇÃO** com sub-query
  - Permite executar um processamento sobre uma tabela e colocar o resultado como tuplas em outra tabela.
- Sintaxe:

```
INSERT   INTO   <Tab>   [  (Campo1, Campo2, ...)  ]
    (SELECT  [* | Campo1, Campo2, ..., Campo n]
        FROM <tab2>  [WHERE <condição>]);
```

- Exemplo:

```
INSERT  INTO  dept
   (SELECT * FROM departamentos
                WHERE cpdepto IN ('A11','A12'));
```

Atualização: DELETE

- Sintaxe:

```
DELETE FROM <Tab> [WHERE <condição>];
```

- Exemplo:

```
DELETE  FROM  ClienteBOM
        WHERE ClienteBOM.Limite < 1500;
```

Atualização: DELETE de tabelas complexas

- Teste:
- Se quisermos identificar e deletar tuplas resultantes de uma junção?
  - Quero deletar todos as tuplas de uma tabela (d1) que tem relacionamento com outra (d2)...
  - Vamos contruir duas tabelas d1 e d2 derivadas de funcionários
- Podemos escrever

```
DELETE d1 FROM d1, d2
        WHERE
          d1.cpfuncionario =
          d2.cpfuncionario;
```

Atualização: UPDATE

- Sintaxe:

```
UPDATE   TABELA
    SET  Atrib1=Valor1,
         Atrib2 = Valor2,
         Atrib3= Valor3
   [ WHERE   condição] ;
```

- Exemplo:

```
UPDATE   Alunos
     SET
          Nome="José Antonio da Silva"
      WHERE
          Alunos.Matricula ="9871234";
```

Atualização: UPDATE sobre múltiplas tabelas

- E se... Temos uma tabela com notas fiscais e outra com seus respectivos itens e PRECISAMOS recalcular o valor total das notas somando os respectivos valores totais de cada item correspondente, como fazer?
- Em outras palavras o UPDATE devera atualizar uma tabela com o resultado de processamento de função de outras tabelas, como fazer?
- O princípio do raciocínio é este:
  - CRIE SUB-QUERIES COM O MAIS BAIXO GRAU DE GRANULARIDADE
  - DEFININDO: GRANULARIDADE
    nível de detalhamento dos dados apresentados em um banco de dados.
    - EM GERAL aparecem em tabelas de desmembramentos N:M, pois quase sempre tem a estrutura de Produtos Cartesianos (que mostram as possibilidades de combinação de Dados).
- Na situação acima o mais baixo nível de granularidade está em itens das notas fiscais... Portanto, devemos processar as totalizações nos itens de notas e gerar os dados que devem ser 'levados' para as notas respectivas.

Atualização: UPDATE sobre múltiplas tabelas

- Então podemos escrever:
- select nu_nota_venda, SUM(qt_vendida*vl_unitario_venda) as vtot
  FROM notas_fiscais_vendas_itens
  GROUP BY nu_nota_venda
  ORDER BY nu_nota_venda

  para totalizar os itens das notas (note o que a projeção gera)
- E para atualizar a tabela com as notas fiscais...
-
```
UPDATE notas_fiscais_vendas,
       ((select nu_nota_venda,
                SUM(qt_vendida*vl_unitario_venda) as vtot
                FROM notas_fiscais_vendas_itens
                GROUP BY nu_nota_venda
                ORDER BY nu_nota_venda) AS vtotitem)
       SET vl_total_nota_venda=vtot
       WHERE notas_fiscais_vendas.nu_nota_fiscal_venda =
             vtotitem.nu_nota_venda;
```

Um Exemplo de Aplicação

ProgramaEntidade (PE)

NomePrograma	CodigoEntity
Pfolha01	0110100

Entidades (E)

CodEntidade	NomedaEntidade
0110100	Funcionários

Programas (P)

CdPrograma	NomedoPrograma
pFolha01	Cadastro de Funcionários

- Para estas tabelas pergunta-se:

a) Quais são os nomes dos programas que acessam a entidade 'FUNCIONARIOS'

Um Exemplo de Aplicação

- Para estas tabelas pergunta-se:

a) Quais são os nomes dos programas que acessam a entidade 'FUNCIONARIOS'

Resolvendo:

Seleção	T ← E [NomeDaEntidade='Funcionários']
Junção	R ← T [CodEntidade=CodigoEntity]PE
Junção	S ← R [NomePrograma=cdPrograma]P
Projeção	F ← S [NomeDoPrograma]

Um Exemplo de Aplicação

- Em SQL teremos:

- CREATE TABLE T AS
  (SELECT * FROM E WHERE NomeDaEntidade='Funcionários')
- CREATE TABLE R AS
  (SELECT * FROM T,PE WHERE T.CodEntidade=PE.CodigoEntity)
- CREATE TABLE S AS
  (SELECT * FROM R,P WHERE R.NomePrograma=P.cdPrograma)
- CREATE TABLE F AS
  (SELECT NomeDoPrograma FROM S)
- DROP TABLE T, R, S

Alterando a Estrutura de uma tabela

- Comando: Alter Table
- Pode ser usado com 3 propósitos:
  - Alterar a estrutura de uma tabela
  - Definir uma chave primária (Primary Key ou PK)
  - Definir uma chave estrangeira – Foreign Key ou FK; (caracterizando o comportamento da FK em função da PK correspondente de outra tabela).

Alterando a Estrutura de uma tabela

- Alter Table
  - Sintaxe:
    ALTER TABLE <TAB>
    < ADD | DROP > <TIPO> [TAM(n)]
    [BEFORE | AFTER <campo>]
  - Exemplo:
    ALTER TABLE funcoes
    ADD teste INT(3)
    NULL AFTER IDFuncao

Alter Table

- Trocando o nome de um campo, a posição na tabela e outras características:

```
ALTER TABLE <tab>
CHANGE COLUMN <NomeAntes> <NomeDepois>
                    <Definições do campo – Tipo+Tam...>
                    <Posição: FIRST|
                              AFTER<campo>|before<campo>|
                              LAST>;
```

- Exemplo:

```
ALTER TABLE enfermeiros
CHANGE COLUMN numcasaenf numcasaenf
                     INT(5) NOT NULL
                    COLLATE 'latin1_swedish_ci'
                     AFTER endenf,
CHANGE COLUMN foneenf foneenf
                    CHAR(12) NOT NULL
                    COLLATE 'utf8_bin'
                    AFTER cidadeenf;
```

Alter Table

- Trocando a posição de um campo de uma tabela

```
ALTER TABLE funcionarios
        MODIFY COLUMN tx_sobre_nome
                      varchar(15)
                      COLLATE utf8_bin NOT NULL
        AFTER tx_iniciais_medias;
```

Alterando a Estrutura de uma tabela

- Alter Table
  - Criando (excluindo) uma Chave Primária
    - Sintaxe:
      ALTER TABLE <TAB>
      [ADD | DROP] PRIMARY KEY (<nome>)
    - Exemplo:
      ALTER TABLE project
      ADD PRIMARY KEY(PROJNO)

Alterando a Estrutura de uma tabela

- O SQL:1992 tem comando que permite criar Chaves Estrangeiras e indicar ao SGBD que faça o controle da Integridade referencial.
- Lembrando da 2ª Regra de Integridade do Mod. Relacional:
  - A Chave Estrangeira pode assumir dois valores:
    - SER NULO
    - Um dos valores da Série de Valores onde é Chave Primária.
  - Sendo assim considera-se dois momentos críticos
    - a remoção de uma linha de uma tabela que contém uma chave primária e/ou
    - a alteração de um valor de uma chave primária.

Alterando a Estrutura de uma tabela

- Alter Table:
  - ALTER TABLE <TAB>
    ADD FOREIGN KEY (<CampoDaTab>)
    REFERENCES <TAB da CP> (<CAMPO CP>)
    ON DELETE [cascade |
    set null |
    no action |
    restrict]
    ON UPDATE [cascade |
    set null |
    no action |
    restrict];

Alterando a estrutura de uma tabela

- ALTER TABLE <TAB>
  ADD FOREIGN KEY (<CampoDaTab>)
  REFERENCES <TAB da CE> (<CAMPO CE>)
  ON DELETE [cascade |set null | no action | restrict]
  ON UPDATE [cascade |set null | no action | restrict];

- cascade – a ação sofrida na CP é propagada para as linhas da CE
- set null – muda o valor da CE para NULO se a CP for excluída
- no action – não faz nada com o valor da CE
- restrict – não permite a alteração na linha onde está a CP

- Alterando a estrutura de uma tabela
  - Pré-requisito para montar uma CE.
  - Os campos das tabelas envolvidas devem ter índices associados às Chaves Primárias e Estrangeiras.

- Atividade:
  - Criem as CPs das tabelas da base exemplo_SQL
  - Criem os índices dos campos correspondentes à CE
  - Criem a integridade referencial mantida no BD com o comando alter table (ou usem o PHPMyADMIN).

Lembretes

- Usamos os comandos SQL para:

- Recuperar dados das linhas de tabelas
- Recuperar dados de colunas específicas

- Mas também podemos...
- Processar dados que estão armazenados em tabelas e **depois** recuperá-los.
- Processar dados de uma tabela e gravá-los em outra tabela.

Trabalhando com Junções

- SELECT * FROM funcionarios, departamentos
  WHERE
  funcionarios.cpdepto=departamentos.cpdepto
- CRIE UMA TABELA dept A PARTIR DE departamentos
- EXCLUA METADE DE SUAS LINHAS
- REPITA O PRIMEIRO COMANDO COM dept NO LUGAR DE departamentos.

- AGORA ESCREVA:
  - SELECT * FROM funcionarios LEFT JOIN dept ON funcionarios.cpdepto=dept.cpdepto
- ANALISE O COMPORTAMENTO DO LEFT JOIN

Trabalhando com Junções

- Teste:

```
SELECT projetos.*,tarefas.*
    FROM tarefas, tarefas_projetos, projetos
    WHERE tarefas.cptarefa= tarefas_projetos.cptarefa AND
            tarefas_projetos.cpprojeto=projetos.cpprojeto
```

- Observe que os campos da tabela tarefas_projetos NÃO são projetados
- Teste:

```
SELECT projetos.*,tarefas.*
    FROM tarefas, tarefas_projetos, projetos
    WHERE tarefas.cptarefa= tarefas_projetos.cptarefa AND
            tarefas_projetos.cpprojeto=projetos.cpprojeto
    ORDER BY projetos. cpprojeto ASC
```

Trabalhando com Junções

- Teste:

```
SELECT projetos.*,COUNT(tarefas_projetos.cptarefa) AS Num_Tarefas
        FROM projetos, tarefas_projetos
        WHERE tarefas_projetos.cpprojeto=projetos.cpprojeto
        GROUP BY projetos.cpprojeto
        ORDER BY projetos.cpprojeto asc
```

Trabalhando com Junções

- Teste (um relacionamento reflexivo):

```
SELECT dsup.cpdepto as cpdepsup,
       dsup.tx_nome_depto as tx_nomsup,
       dsup. cpfuncionario_gerente as gerente,
       dsub.cpdepto as cpdepsub,
       dsub.tx_nome_depto as tx_nomsub
       FROM departamentos as dsup,
            departamentos as dsub
       WHERE
            dsup.cpdepto =
            dsub.cpdepto_superior
```

- Responde: Quais são os departamentos subalternos de cada departamento superior?

Trabalhando com Junções

- Teste:

```
SELECT f1.cpfuncionario as cpchefe,
       f1.tx_primeiro_nome as tx_prinome_chefe,
       f1.nu_ramal as nu_ramal_chefe,
       d1.cpdepto, d1.tx_nome_depto,
       f2.cpfuncionario,
       f2.tx_primeiro_nome, f2.nu_ramal
       FROM funcionarios as f1,
            departamentos as d1,
            funcionarios as f2
       WHERE
            f1.cpfuncionario =
            d1.cpfuncionario_gerente AND
            d1.cpdepto=f2.cpdepto
```

- Responde: Quais são os funcionários de cada departamento e seu respectivo gerente?

- Aderência ao Padrão.
  - Os SGBD podem ou não adotar o Padrão.
  - Hoje os SGBD (quase todos) usam o SQL:1992.
  - Alguns têm o SQL:1999, mas têm sintaxe própria para definir Triggers e Stored Procedures.
  - Exercitar o SQL:1992 no MySQL + PostgreSQL

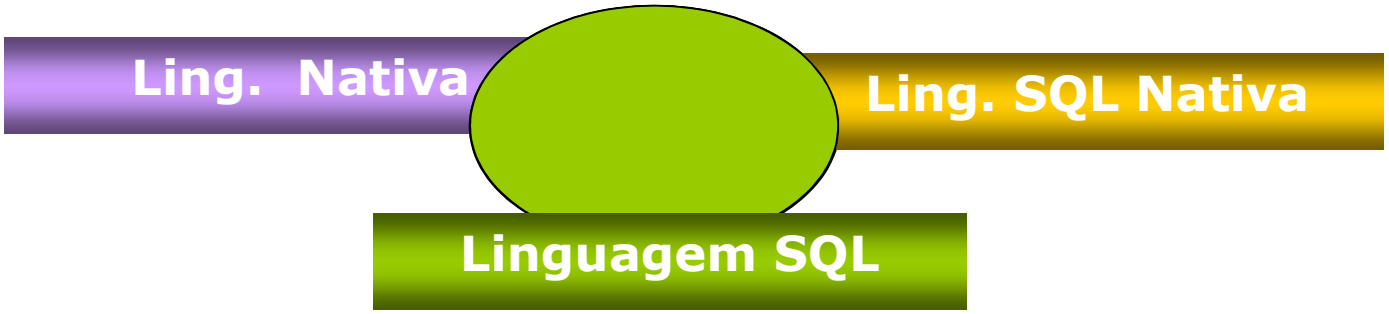
- Aderência ao Padrão

- Vemos que os fabricantes de SGBDs podem implementar mais funções do que as padronizadas (até mesmo colocando mais comandos com performance melhorada). Chamamos de Linguagem NATIVA (ou proprietária);

- Mas não é interessante fugir do Padrão. Então muitos SGBDs "FALAM" SQL como uma alternativa de linguagem. Chamamos de SQL-PADRÃO.

- Aderência ao Padrão

- Outros ainda percebem que podem combinar SQL com algumas funções do seu produto e geram comandos sintaticamente semelhantes com SQL mas com mais potencial de funções. Chamamos de SQL-NATIVO;
- Esquematicamente podemos representar as linguagens assim:



Padronização SQL – Tendências

- A SQL foi um marco na história do desenvolvimento de linguagens de computação;
  - Sabemos como deve ser a linguagem e o que ela deveria fazer;
  - Baseados nos fundamentos teóricos lançados na apresentação do Modelo Relacional;
  - Conseqüência:
    - Linguagem é elegante e econômica, consistindo de relativamente poucos comandos;
    - A simplicidade da linguagem torna-se conveniente tanto para o usuário eventual como para um projetista sofisticado;
- Ainda se desenvolve padronização do SQL para alguns segmentos de linguagens, tais como:
  - gerência de transações melhorada;
  - especificação de regras definida pelo usuário;
  - Especificação de estruturas de tabelas com os tipos de dados padrão.
- Também ocorrem alterações na SQL por sugestões de usuários e como resultados de testes no desenvolvimento da linguagem.

Conclusão

- A SQL é hoje uma linguagem de comunicação padronizada entre os bancos de dados, dado a sua divulgação e amplitude mercadológica;

- Muitos gerenciadores de Banco de Dados adotam a SQL como acessório para sua linguagem nativa já prevendo a interligação de vários gerenciadores e Base de Dados diferentes.