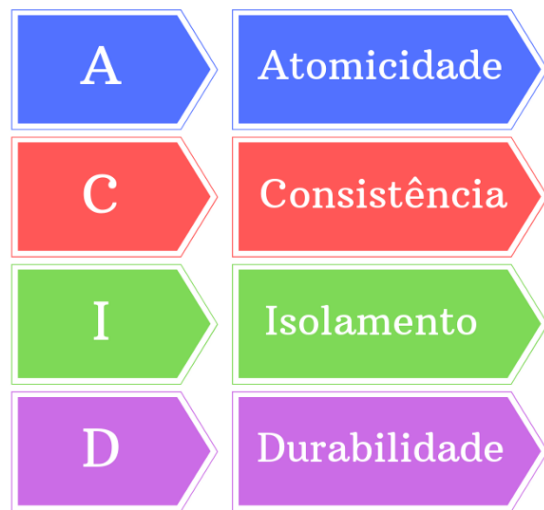
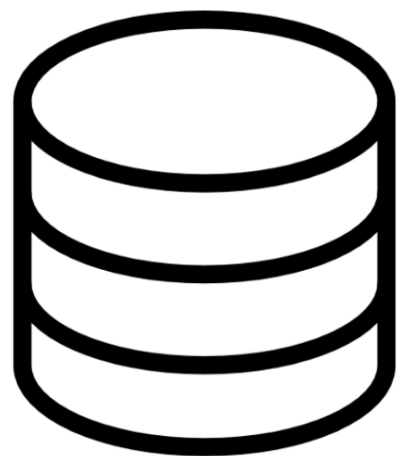


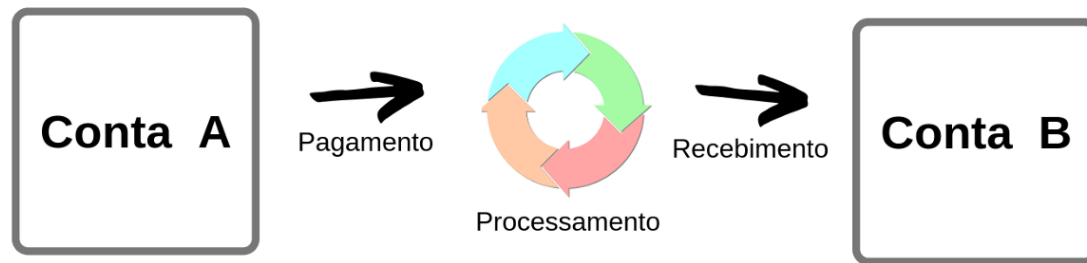
Banco de Dados

Profº Haroldo Gomes

Introdução ao SQL



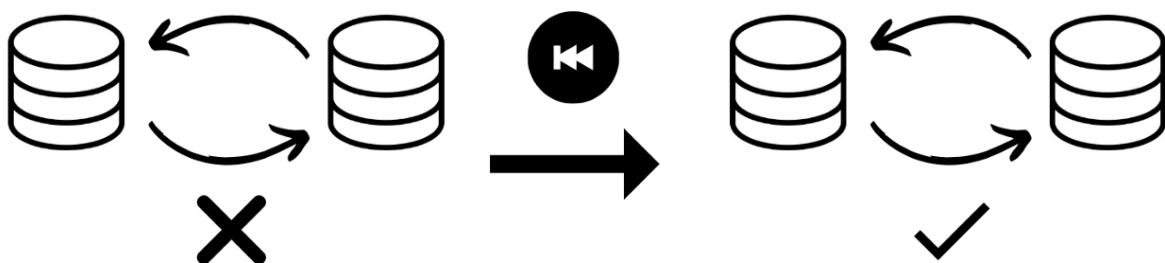
*Antes de
partir para
o SQL ...*



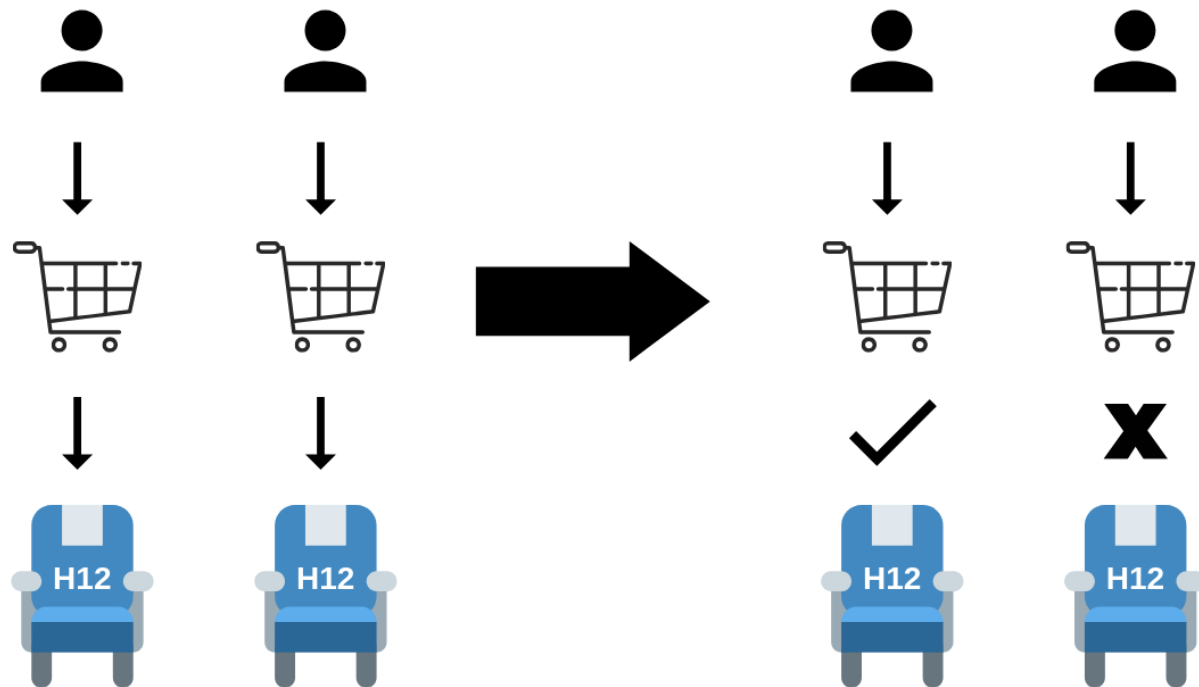
Atomicidade

"Tudo ou nada", que é basicamente, ou todas operações referentes a um assunto funcionam, , ou aquele assunto inteiro falha.

Consistência



A consistência é assegurar que o banco de dados estará em um estado válido, então ao se realizar uma transação, o banco irá de um estado válido para outro também válido.



Isolamento

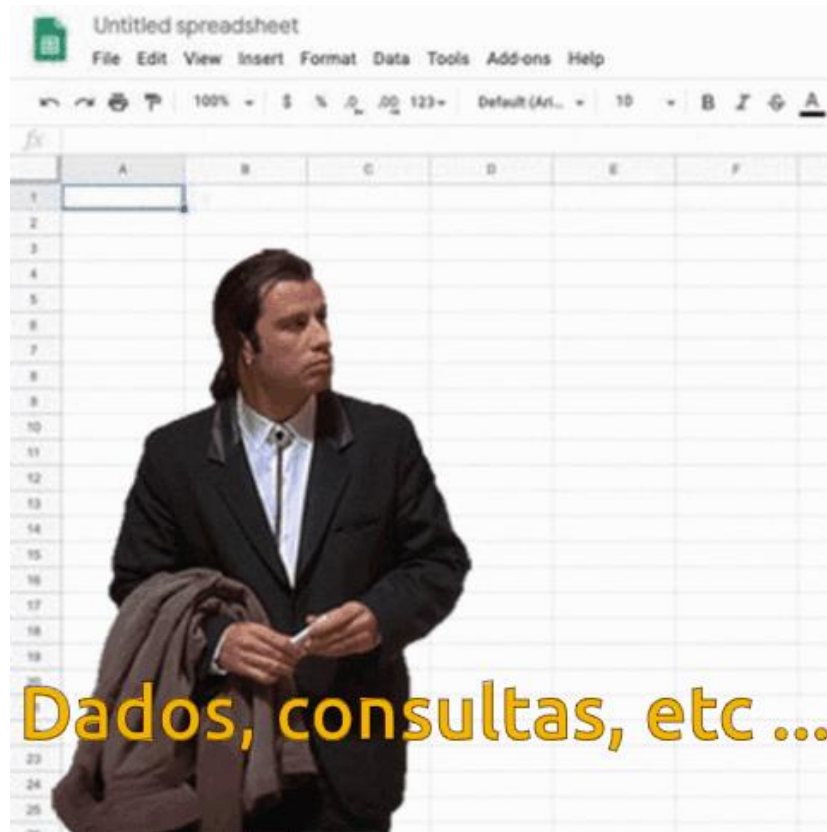
Garante que as transações possam ser feitas de forma simultânea e independente.



Durabilidade

A durabilidade irá garantir que após as transações serem efetivadas (o termo disso é "commit"), em caso de falha ou reiniciamento do sistema os dados não deixarão de existir.

Linguagem SQL

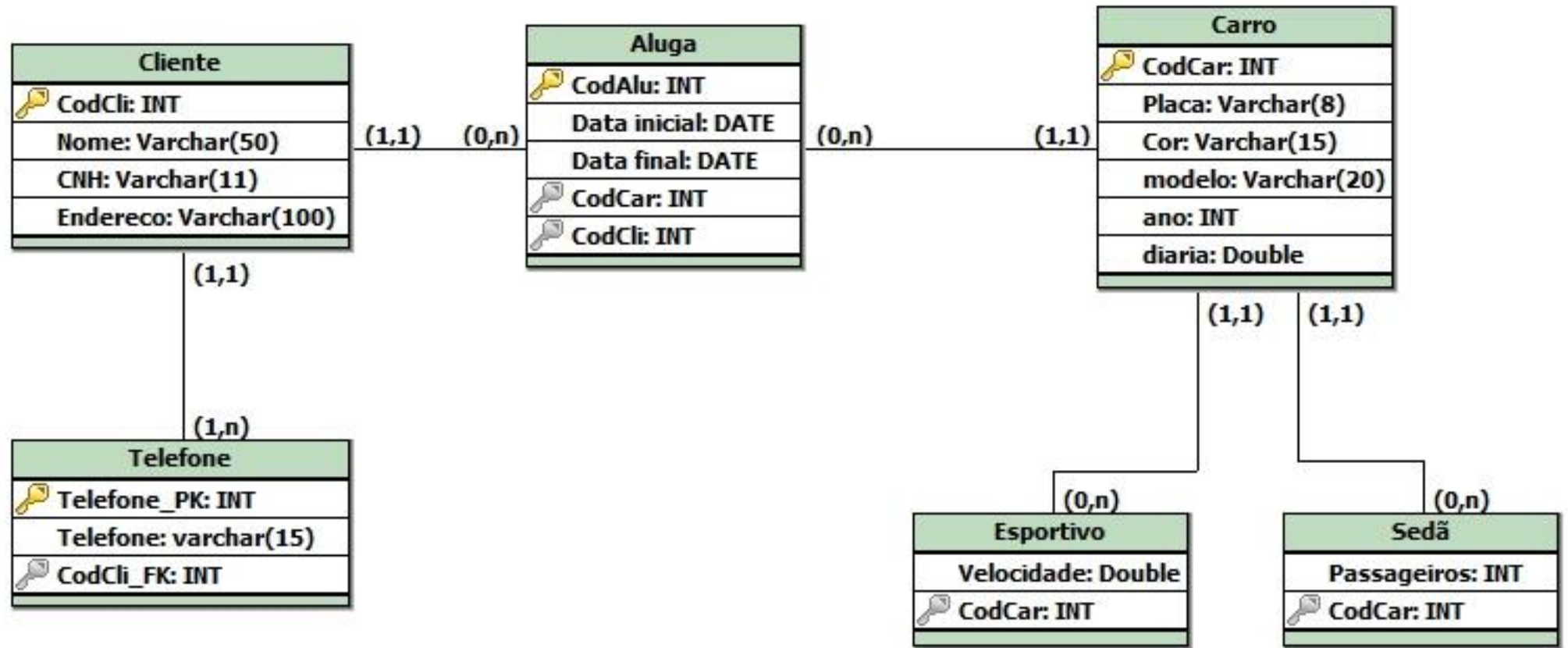


Dados, consultas, etc ...



Linguagem SQL

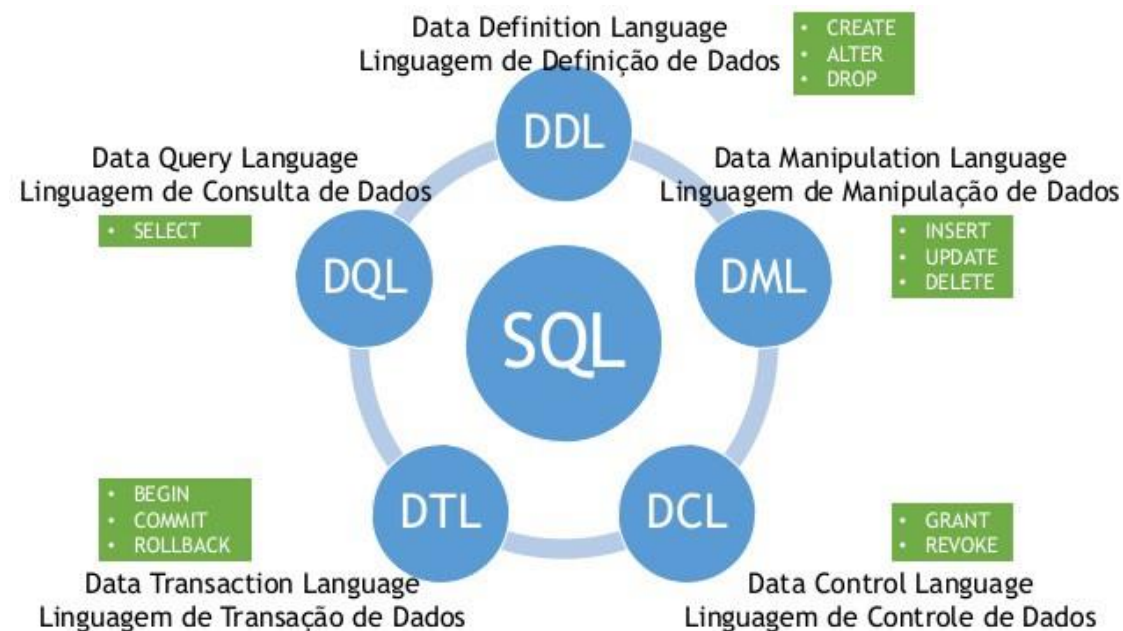
O Modelo Relacional prevê, desde sua concepção, a existência de uma linguagem baseada em caracteres que suporte a definição do esquema físico (tabelas, relacionamentos e restrições) e sua manipulação (inserção, consulta, atualização e remoção).



Linguagem SQL

A Linguagem SQL (*Structured Query Language*) é padrão para SGBDs Relacionais

– padrão ANSI (*American National Standards Institute*)





Conjuntos de Comandos da Linguagem SQL

A Linguagem SQL pode ser dividida em 5 conjuntos de comandos:

- **Recuperação de dados:** comando SELECT
- Linguagem de manipulação de dados (**DML - Data Manipulation Language**): comandos para inserções (INSERT), atualizações (UPDATE) e exclusões (DELETE)



Conjuntos de Comandos da Linguagem SQL

- Linguagem de definição de dados (**DDL - Data Definition Language**): comandos para criação e manutenção de objetos do banco de dados: CREATE, ALTER, DROP, RENAME e TRUNCATE
- Linguagem para **controle de transações**: COMMIT, ROLLBACK e SAVEPOINT
- Linguagem para **controle de acesso a dados**: GRANT e REVOKE



Criando um BD e uma Tabela

```
CREATE DATABASE MinhaCaixa;
```

```
use MinhaCaixa;
```

```
CREATE TABLE Clientes (  
    ClienteCodigo int,  
    ClienteNome varchar(20)  
);
```

```
CREATE TABLE "nome_tabela"  
("coluna 1" "tipo_dados_para_coluna_1",  
"coluna 2" "tipo_dados_para_coluna_2",  
... );
```

Especificando uma Chave primária (em BD's diferentes)

MySQL:

```
CREATE TABLE Customer  
(SID integer,  
Last_Name varchar(30),  
First_Name varchar(30),  
PRIMARY KEY (SID) );
```

Oracle:

```
CREATE TABLE Customer  
(SID integer PRIMARY KEY,  
Last_Name varchar(30),  
First_Name varchar(30) );
```

SQL Server:

```
CREATE TABLE Customer  
(SID integer PRIMARY KEY,  
Last_Name varchar(30),  
First_Name varchar(30) );
```

MySQL:

```
ALTER TABLE Customer ADD PRIMARY KEY (SID);
```

Oracle:

```
ALTER TABLE Customer ADD PRIMARY KEY (SID);
```

SQL Server:

```
ALTER TABLE Customer ADD PRIMARY KEY (SID);
```

Especificando uma chave estrangeira

MySQL:

```
CREATE TABLE ORDERS
(Order_ID integer,
Order_Date date,
Customer_SID integer,
Amount double,
PRIMARY KEY (Order_ID),
FOREIGN KEY (Customer_SID) REFERENCES CUSTOMER (SID));
```

Tabela **CUSTOMER**

Nome da Coluna	Característica
SID	Chave Primária
Last_Name	
First_Name	

Tabela **ORDERS**

Nome da Coluna	Característica
Order_ID	Chave Primária
Order_Date	
Customer_SID	Chave Externa
Amount	

Selecionando um dado

```
SELECT "nome_coluna" FROM "nome_tabela";
```

Tabela *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

```
SELECT Store_Name FROM Store_Information;
```


Inserindo Dados

```
INSERT INTO "nome_tabela" ("coluna 1", "coluna 2", ...)  
VALUES ("valor 1", "valor 2", ...);
```

Tabela *Store_Information*

Nome da Coluna	Tipo de Dados
Store_Name	char(50)
Sales	float
Txn_Date	datetime

```
INSERT INTO Store_Information (Store_Name, Sales, Txn_Date)  
VALUES ('Los Angeles', 900, '10-Jan-1999' );
```

Atualizando um BD

```
UPDATE "nome_tabela"  
SET "coluna 1" = [novo valor]  
WHERE "condição";
```

Tabela *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

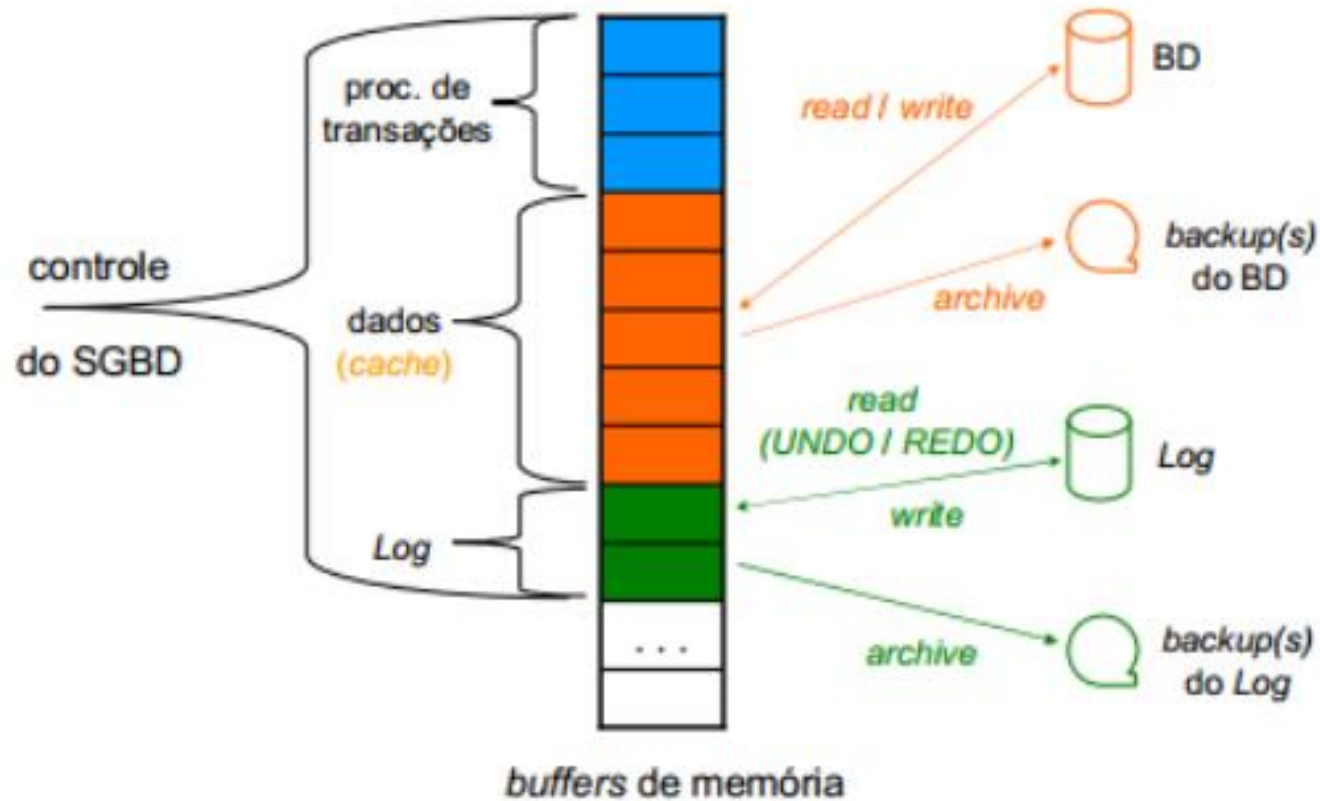
```
UPDATE Store_Information  
SET Sales = 500  
WHERE Store_Name = 'Los Angeles'  
AND Txn_Date = '08-Jan-1999';
```

Inserindo N linhas

```
INSERT INTO "tabela 1" ("coluna 1", "coluna 2", ...)  
SELECT "coluna 3", "coluna 4", ...  
FROM "tabela 2";
```

```
INSERT INTO Store_Information (Store_Name, Sales, Txn_Date)  
SELECT Store_Name, Sales, Txn_Date  
FROM Sales_Information  
WHERE Year (Txn_Date) = 1998;
```

Recuperação de Informação





Recuperação ...

O *buffering* de páginas de disco (blocos) do banco de dados no cache de memória principal do SGBD.

O *caching* de dos blocos sobre o controle do SGBD, independente do sistema operacional.

Utilizando o 'Select Distinct'

```
SELECT DISTINCT "nome_coluna"  
FROM "nome_tabela";
```

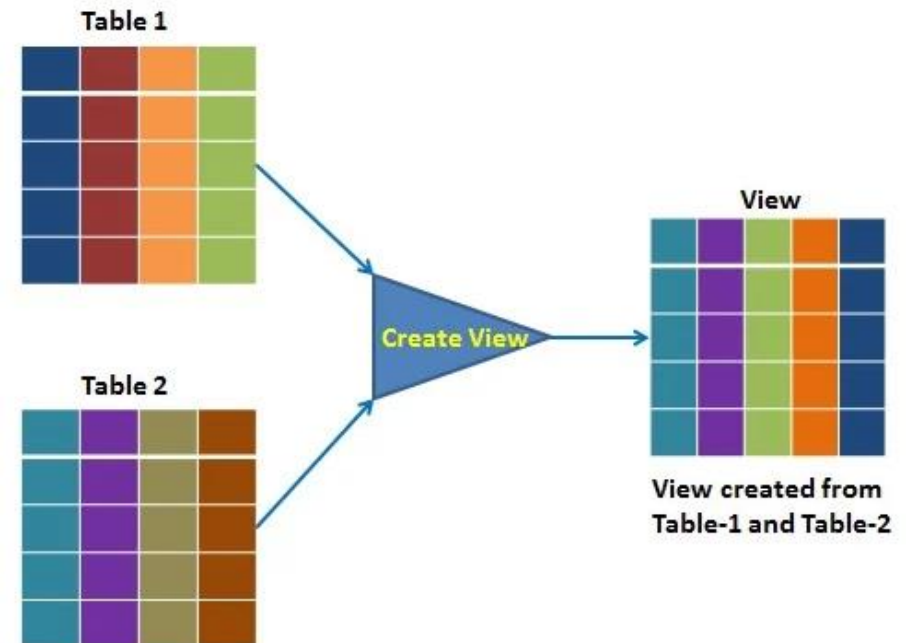
Tabela *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

```
SELECT DISTINCT Store_Name FROM Store_Information;
```

Criando Views

A **view** pode ser definida como uma tabela virtual composta por linhas e colunas de dados vindos de tabelas relacionadas em uma query (um agrupamento de SELECT's, por exemplo). As linhas e colunas da view são geradas dinamicamente no momento em que é feita uma referência a ela.





Vantagens

Reuso: as views são objetos de caráter permanente. Pensando pelo lado produtivo isso é excelente, já que elas podem ser lidas por vários usuários simultaneamente.

Segurança: as views permitem que ocultemos determinadas colunas de uma tabela. Para isso, basta criarmos uma view com as colunas que achamos necessário que sejam exibidas e as disponibilizarmos para o usuário.

Simplificação do código: as views nos permitem criar um código de 'programação' muito mais limpo, na medida em que podem conter um SELECT complexo. Assim, **criar views** para os programadores a fim de poupá-los do trabalho de criar SELECT's é uma forma de aumentar a produtividade da equipe de desenvolvimento.



Criando uma View

```
CREATE VIEW nome_da_view AS SELECT * FROM nome_tabela;
```

Executando a *Query*

```
SHOW TABLES;
```

Alterando uma View

```
ALTER VIEW nome_da_view AS SELECT * FROM nome_outra_tabela;
```

Excluindo uma View

```
DROP VIEW nome_da_view;
```



E quando o BD escala "demais"?

Arquitetura de dados

Escalabilidade

Padrões em consultas

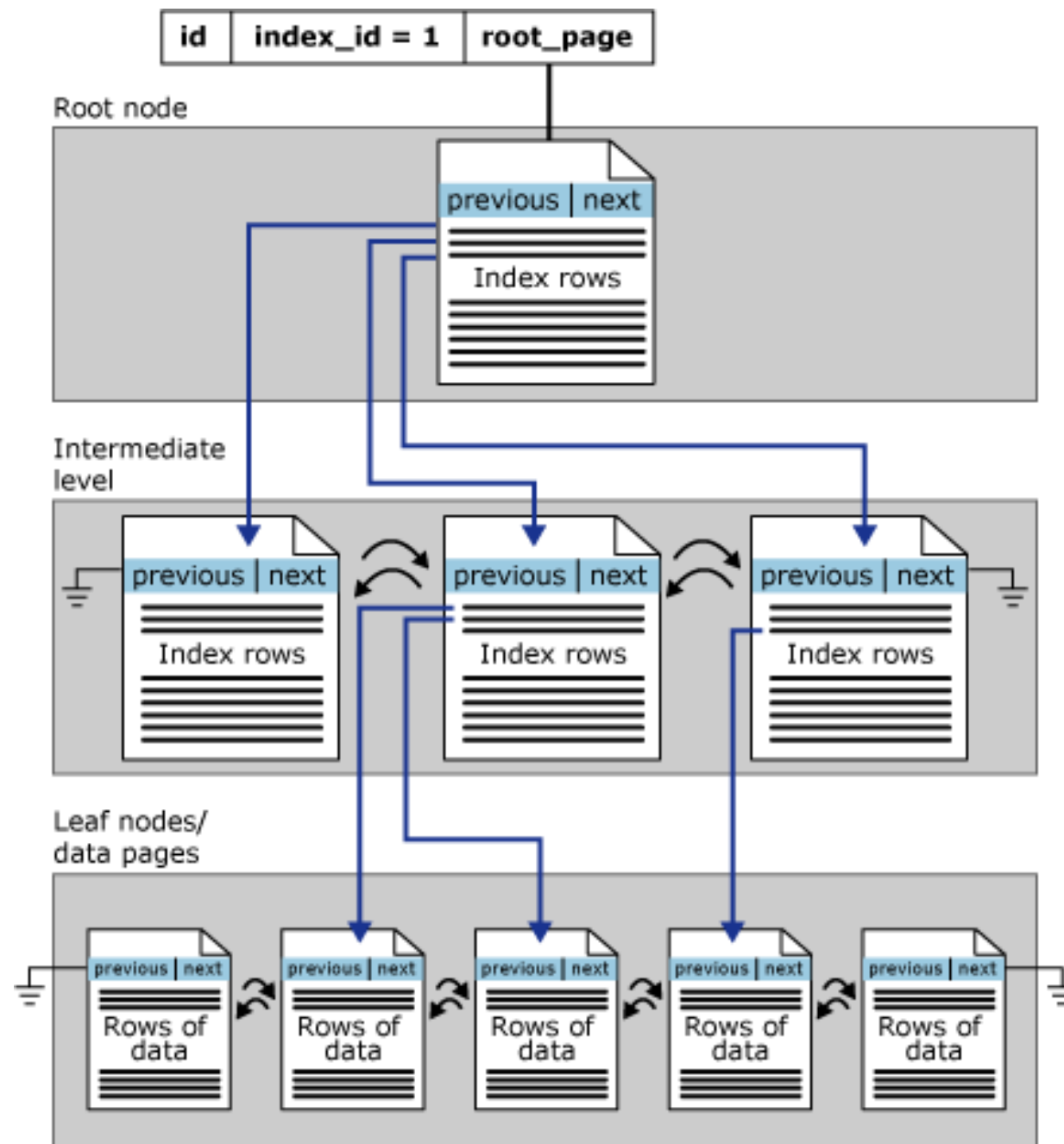
Quando temos uma tabela e executamos uma operação de SELECT sobre ela, filtrando por um ou vários campos, o gerenciador do banco efetua uma ação chamada "TABLE SCAN". Essa ação consiste em percorrer toda a tabela, avaliando cada registro. Caso o registro atenda às condições definidas no filtro, ele é incluído no conjunto de retorno, senão, é apenas desconsiderado.

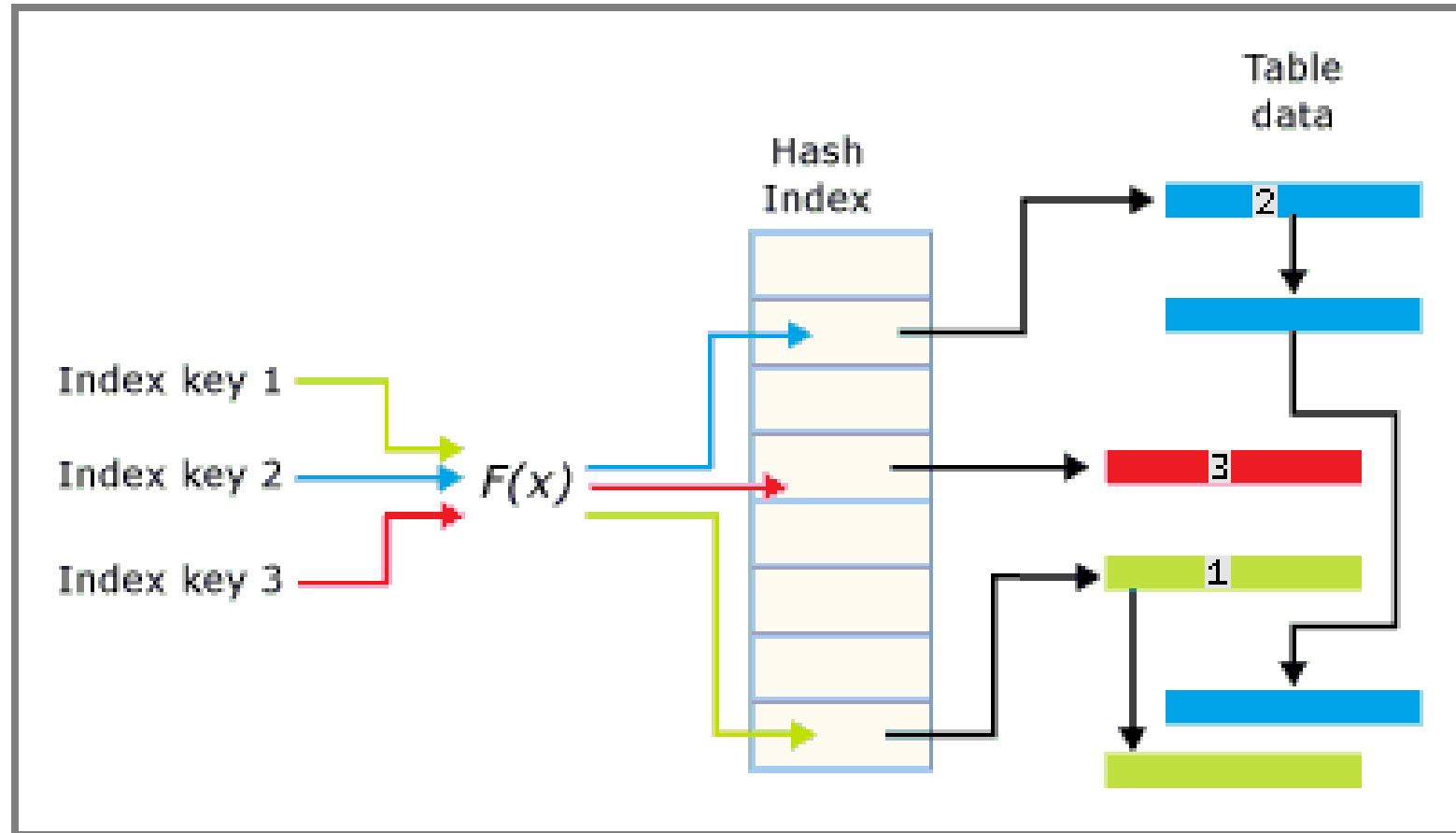
		<table><tr><th>Codigo</th><th>Nome</th></tr><tr><td>1</td><td>Joao</td></tr><tr><td>2</td><td>Antonio</td></tr><tr><td>3</td><td>Maria</td></tr></table>	Codigo	Nome	1	Joao	2	Antonio	3	Maria
Codigo	Nome									
1	Joao									
2	Antonio									
3	Maria									
Codigo = 3? Não. Desconsidere.	→									
Codigo = 3? Não. Desconsidere.	→									
Codigo = 3? Sim. Retornar registro.	→									

Como solucionar e otimizar o processo ? 🧐



Criando Index (índices) - Otimização de Consultas





Quando criamos um índice em uma coluna, o gerenciador do banco ordena a tabela por essa coluna e a partir de então os filtros (sobre essa coluna) são feitos através de uma busca binária.

Codigo	Nome
1	Joao
2	Antonio
3	Maria
4	Carlos
5	Francisca
6	Pedro
7	Jose
8	Maria
9	Xavier
10	Joaquim

} Primeira metade

} Segunda metade



Criando 'Index'

```
CREATE TABLE CLIENTES  
(  
    Codigo INT,  
    Nome VARCHAR(50),  
    INDEX (Codigo)  
);
```


Utilizando a Ordenação

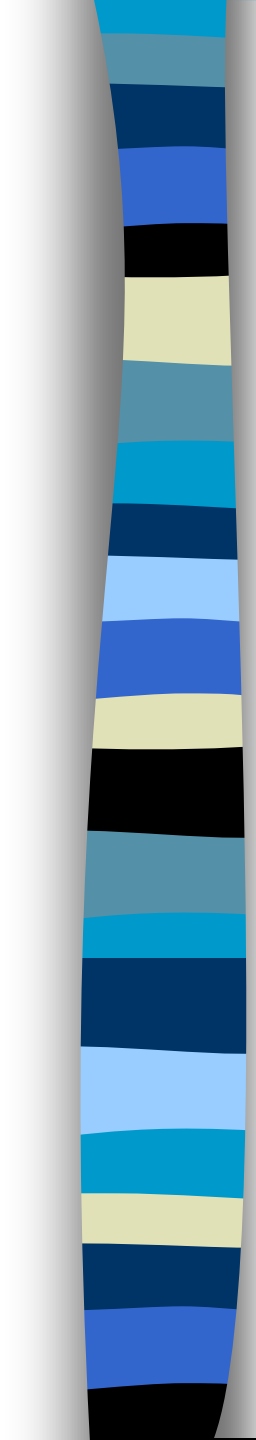
ORDER BY Clause in SQL

EmployeeID	EmployeeLastName	EmployeeFirstName	EmailID
003	Jones	Amy	amy@gmail.com
006	Brown	Dan	dan@gmail.com
001	Donald	Jo	jo@gmail.com

SELECT *
FROM Employee
ORDER BY
EmployeeLastName;

Result

EmployeeID	EmployeeLastName	EmployeeFirstName	EmailID
006	Brown	Dan	dan@gmail.com
001	Donald	Jo	jo@gmail.com
003	Jones	Amy	amy@gmail.com

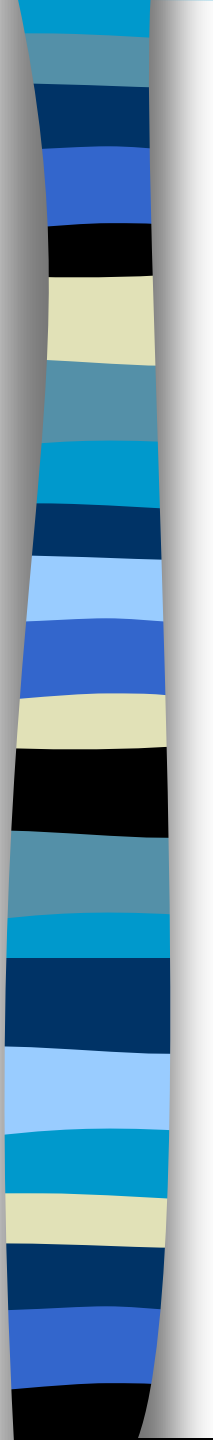


```
SELECT "nome_coluna"  
FROM "nome_tabela"  
[WHERE "condição"]  
ORDER BY "nome_coluna" [ASC, DESC];
```

Tabela ***Store_Information***

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
San Francisco	300	08-Jan-1999
Boston	700	08-Jan-1999

```
SELECT Store_Name, Sales, Txn_Date  
FROM Store_Information  
ORDER BY Sales DESC;
```



Agrupando dados

```
SELECT "nome_coluna1", SUM("nome_coluna2")  
FROM "nome_tabela"  
GROUP BY "nome_coluna1";
```

Tabela *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

```
SELECT Store_Name, SUM (Sales)  
FROM Store_Information  
GROUP BY Store_Name;
```



<u>Store_Name</u>	<u>SUM (Sales)</u>
Los Angeles	1800
San Diego	250
Boston	700