serratec
Parque Tecnológico Região Serrana

Banco de Dados

SUMÁRIO

1. Conceitos Gerais sobre Bancos de Dados

Os bancos de dados desempenham um papel crucial em quase todas as áreas nas quais são necessários usos de computadores, desde o uso de bancos simples como em um e-commerce, na engenharia, medicina, genética até mesmo big datas como mecanismos de buscas, informações geográficas e redes sociais . O termo database (banco de dados) é tão utilizado que se faz necessário uma definição conceitual do que é um dado para entendermos o que é de fato de um banco de dados.

Dados são valores atribuídos a algo. Estes valores não precisam ser necessariamente números. Dados são códigos que constituem a matéria prima da informação, ou seja, é a informação não tratada. Os dados representam um ou mais significados que isoladamente não podem transmitir uma mensagem ou representar algum conhecimento.

Um banco de dados é uma coleção de dados que se relacionam. Em outras palavras, são fatos conhecidos que podem ser registrados e possuem significado implícito. Por exemplo: um nome, telefone e e-mail de uma pessoa conhecida podem ser registrados num smartphone, que possui um banco de dados simplificado em seu software. Essa coleção de dados relacionados é um banco de dados.

Podemos dizer que um banco de dados possui as seguintes propriedades:

- Um banco de dados representa algum aspecto do mundo real, chamado de universo de discurso ou mini mundo. As mudanças nesse mini mundo são reproduzidas no banco de dados;
- Um banco de dados é uma coleção de dados coerentes com algum significado. Uma variedade de dados aleatória não pode ser chamado de banco de dados;
- Um banco de dados é pensado, criado e preenchido com dados para uma finalidade específica. Ele possui um grupo de usuários e aplicações prévias nas quais esses usuários estão interessados.

Em suma, um banco de dados tem uma fonte de dados originários, que se relacionam com eventos do mundo real e possuem um público interessado em seu conteúdo. Os usuários finais podem realizar ações que geram alterações nesse banco, como por exemplo: um registro de um funcionário que mudou de endereço. Tal mudança deverá ser refletida nesse banco. Para que tal banco de dados seja considerado confiável e preciso é necessário que tais modificações sejam refletidas o mais rápido possível.

Um banco de dados pode ser criado e mantido manualmente ou pode ser computadorizado. Por exemplo: um almoxarifado em que o solicitante preenche um formulário físico do item que foi retirado. Um banco de dados computadorizado pode ser criado e mantido por um grupo de programas para determinada tarefa ou por um sistema de gerenciamento de banco de dados (SGBD).

Um SGBD é um sistema que possibilita o processo de *definição, construção, manipulação e compartilhamento* de banco de dados entre usuários e aplicações. A definição de um banco consiste em especificar os tipos de dados, as estruturas e restrições dos dados que serão armazenados. A **definição** é armazenada pelo SGBD na forma de um dicionário, chamado de metadados. A **construção** do banco é o processo de armazenar os

dados em algum meio controlado pelo SGBD. A **manipulação** de um banco de dados abrange consultas, atualizações e geração de relatórios a partir dos dados. O **compartilhamento** propicia que vários usuários e programas acessem simultaneamente o banco de dados.

A figura 1 ilustra os conceitos abordados até aqui.

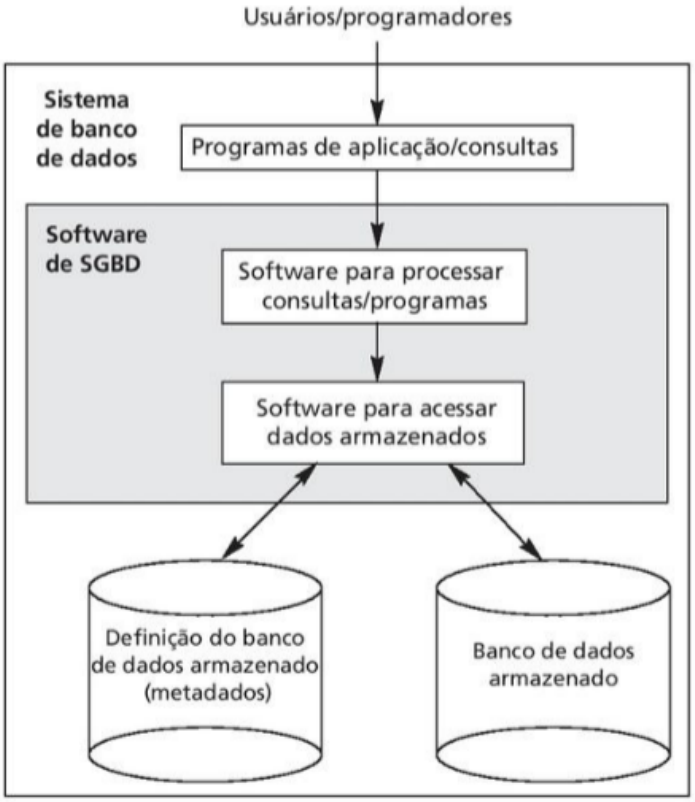

Figura 1 - Diagrama de um sistema de banco de dados (Puga, Sandra. 2013).

2. Modelo Relacional

O modelo relacional foi criado por Edgar F. Codd, nos anos 70 e começou a ser utilizado no início dos anos 80. A ideia do modelo baseia-se no preceito de que as informações em uma base de dados podem ser consideradas como relações matemáticas e que podem ser representadas através do uso de tabelas, de maneira padronizada, onde as linhas representam os eventos de uma entidade e as colunas representam os atributos de uma entidade do modelo conceitual.

As relações no modelo relacional são conjuntos de dados vistos como tabelas cujas operações são baseadas na álgebra relacional (projeção, produto cartesiano, seleção, junção, união e subtração) e que manipulam um conjunto de dados ao invés de um único registro. Em outras palavras, cada operação realizada afeta um conjunto de linhas e não

apenas uma única linha, ainda que algumas operações possam afetar uma única linha (conjunto com um único elemento).

Do mesmo modo, os retornos das consultas são sempre na forma de uma tabela. As operações são executadas por linguagens de alto nível, sendo a SQL a linguagem padrão para os bancos de dados relacionais e universalmente usada, padronizada pelo ANSI (American National Standard Institute).

Entre as principais vantagens do modelo relacional estão a independência total dos dados, a visão múltipla, maior segurança no acesso aos dados armazenados, a facilidade e agilidade tanto para a consulta como para a alteração, etc.

2.1 Modelo Relacional e o Conceito de Chaves

As chaves são regras que determinam o comportamento sobre uma tabela, sendo estas bases para estabelecer os relacionamentos.
Vamos abordar os três tipos de chaves: *Chave Primária (Primary Key - PK), Chave Estrangeira (FOREIGN KEY - FK) e Chave Candidata (Alternativa).*

Chave Primária (Primary Key - PK) - uma chave primária é um regra implementada em uma coluna ou em um conjunto de colunas que garante que os valores contidos nesta colunas sejam únicos, ou seja, não se repetem. Por exemplo: num cadastro de pessoa física, o CPF será a chave primária, levando em conta que é um atributo único, que não se repete.
Em uma tabela, pode-se criar chaves primárias compostas, para que um evento nunca se repita num mesmo momento. Por exemplo: Aluguel de Carro - podemos garantir que um mesmo cliente nunca conseguirá locar o mesmo carro na mesma data e no mesmo horário.
Chave Estrangeira (FOREIGN KEY - FK) - uma chave estrangeira é formada por atributos que são chave primária em outra tabela, deste modo, ela serve para estabelecer relacionamentos entre as tabelas do um banco de dados. Quando duas tabelas estão relacionadas através de uma coluna, devemos observar que em uma tabela esta coluna será chave primária e na outra tabela, a chave estrangeira, fazendo assim a ligação (relacionamento) entre as duas tabelas.
Chave Candidata (Alternativa) - A chave candidata é formada por um atributo que identifica uma única linha (tupla) na tabela. Como uma tabela pode possuir mais de um atributo de identificador único, podemos ter várias chaves candidatas em uma mesma tabela, sendo que apenas uma das chaves candidatas pode ser escolhida para ser a chave primária da tabela. As demais permanecem como chaves candidatas. Ex.: Uma tabela ALUNOS tem uma chave candidata formada pelo CPF e que possui a coluna matrícula como chave primária. Ao efetuar uma consulta, ambas identificaram uma única linha na tabela ALUNOS, porém a chave matrícula é a primária e a chave CPF é candidata.

Podemos dizer que: uma tabela deve ser acessível por qualquer coluna, mesmo que não tenha sido definida como chave; O relacionamento entre duas tabelas é lógico e representado através de chaves estrangeiras; É efetuado o uso de linguagens não-procedurais e auto-contidas; (SQL) .

3. Modelagem Entidade - Relacionamento

Um modelo conceitual de dados é um modelo de dados de alto nível. Sua principal finalidade é capturar os requisitos de informação e regras de negócio sob o ponto de vista do negócio. Isto é, um modelo que não sofre interferência de fatores tecnológicos e fatores de projeto em sua construção e de fácil entendimento das partes interessadas que não atuam na área da tecnologia. No desenvolvimento de soluções é o primeiro modelo que deve ser desenvolvido, já na fase de levantamento de requisitos.

3.1. O uso de modelos de dados conceituais de alto nível

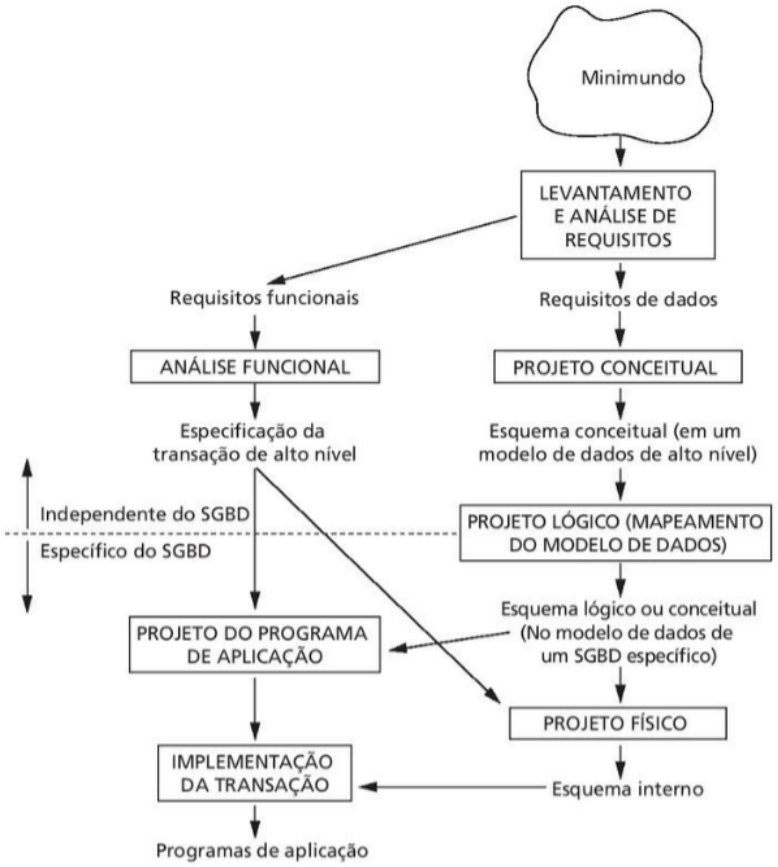

Figura 2. Um diagrama simplificado com as principais fases do projeto de banco de dados (Puga, Sandra. 2014)

Na figura acima é mostrado um diagrama simplificado do processo de um projeto de banco de dados. A primeira fase é o levantamento e análise de requisitos. Nesta etapa, é realizada a entrevista dos usuários interessados para que seja entendido e documentado os requisitos de dados. Com esta etapa concluída têm-se um conjunto de requisitos dos usuários de forma objetiva. A partir daí, tais requisitos devem ser especificados de forma mais completa e detalhada possível. Paralelamente, deve-se determinar os requisitos

funcionais da aplicação. Ou seja, as operações definidas pelo usuário que serão aplicadas ao banco, incluindo atualizações e consultas.

Após os requisitos levantados e analisados, inicia-se a criação de um esquema conceitual, usando um modelo de dados de alto nível. Essa etapa chama-se projeto conceitual. O esquema conceitual é uma descrição sucinta dos requisitos de dados dos usuários com a inclusão de alguns detalhes dos tipos de entidades, restrições e relacionamentos; Esses conceitos geralmente são mais fáceis de entender e auxiliam na comunicação com os usuários não técnicos. Desta forma, essa abordagem garante que os requisitos de dados dos usuários sejam atendidos e não estejam em conflito, e como não há qualquer preocupação com a implementação ou armazenamento, facilita na criação do projeto de banco de dados conceitual.

Simultaneamente ou logo após o esquema conceitual, as operações básicas do modelo de dados podem ser usadas para especificar as consultas e operações do usuário identificadas durante a análise funcional. Isso também serve para a validação se o esquema conceitual atende a todos os requisitos identificados.

A etapa seguinte no projeto é a implementação real do próprio banco de dados, através de um SGBD com o modelo relacional (SQL)-, transformando o modelo de dados de alto nível para o modelo de dados da implementação. Chamamos esta etapa de projeto lógico ou mapeamento do modelo de dados.

4. Modelagem Física

A fase do projeto físico é a última etapa, na qual as estruturas de armazenamento internas, organizações de arquivo, índices, caminhos de acesso e parâmetros físicos do projeto são especificados. Paralelamente, os programas de aplicação são projetados e implementados como transações de banco de dados correspondentes às especificações da transação de alto nível. Essa etapa é também conhecida como Modelagem Física. Tal modelagem envolve algumas estruturas, a saber:

- SGBD - Sistema Gerenciador de Bancos de Dados: software responsável pela criação e manutenção de toda a estrutura física relativa a um banco de dados. Há diversas opções disponíveis, tanto comerciais quanto open source, entre as quais podemos destacar: Postgresql e MySQL (open source), SQL Server e Oracle (comerciais);
- Banco de Dados: estrutura de maior hierarquia, responsável por agrupar as tabelas e seus relacionamentos. Normalmente é construído em função de um determinado negócio ou necessidade, sendo recomendado não misturar diferentes negócios ou sistemas relacionados em um mesmo banco de dados;
- Tabela: estrutura responsável por armazenar os dados. É composta por colunas e se relaciona com outras tabelas;
- Colunas: tipo de estrutura mais atômica, responsável por armazenar um dado específico, como, por ex: nome, CPF, etc. É composta por um tipo de dado,

que diz respeito ao dado a ser armazenado, ou seja, se é um dado do tipo numérico, decimal, textual, etc. - e também por uma quantidade específica de registros/caracteres (dependente do tipo de dado);

- Relacionamentos: diz respeito ao tipo de relacionamento que pode ser estabelecido entre duas tabelas. Ainda sobre esse conceito, há outros relacionados, como Cardinalidade, por exemplo.

Sobre o termo mencionado acima, Cardinalidade, cabe destacar as formas de relacionamentos entre tabelas possíveis em um banco de dados relacional:

1. um-para-um: quando um registro em uma tabela possui um único correspondente em outra tabela;
2. um-para-muitos: quando um registro em uma tabela possui mais de um registro correspondente em outra tabela;
3. muitos-para-muitos: quando mais de um registro em uma tabela se relaciona com mais de um registro em outra tabela. Esse tipo de relacionamento precisa de uma atenção especial em sua implementação, no modelo físico, sendo necessário a criação de uma tabela, chamada tabela de ligação, entre as tabelas relacionadas.

Sobre as definições do Modelo Físico, a tabela abaixo mostra a correspondência dos termos entre o modelo relacional, conceitual, e o modelo físico:

Modelo Conceitual	***Modelo Físico***
dado	dado
atributo	coluna (ou campo)
registro (ou tupla)	registro (ou linha)
entidade	tabela

5. Normalização - Formas Normais

O processo de transformação de um modelo conceitual em um modelo físico é realizado, na maioria das vezes, tendo como base a expertise do profissional envolvido em tal processo, assim como suas experiências anteriores e também conhecimentos específicos do negócio. Logo, trata-se, em suma, de uma atividade, de certa forma, subjetiva. Embora possa ser muito eficaz, é importante termos formas de validar a qualidade do projeto de modelo físico de um banco de dados. Para isso podemos utilizar a Normalização, cuja função é garantir a consistência das informações a serem armazenadas no banco de dados. Para isso, com

base nas Formas Normais existentes, são examinados os atributos de uma entidade (colunas da tabela) a fim de se identificar possíveis problemas ou anomalias que possam ocorrer nas operações a serem executadas sobre a base de dados. Vejamos, a seguir, as principais formas normais existentes e como aplicá-las em um modelo conceitual.

5.1. Formas Normais

O fundamento conceitual do processo de normalização tem sua base nas relações matemáticas (mais precisamente na teoria dos conjuntos) e tem como principal objetivo estruturar as entidades e as relações entre as entidades eliminando grupos repetitivos, redundância de dados, dependências parciais entre colunas e chaves-primárias, dependências de atributos multivalorados, entre outros. São três as principais formas normais, sendo essas conhecidas pelas siglas: 1FN, 2FN e 3FN

5.1.1. 1FN

Para que uma tabela possa estar em conformidade com a primeira forma normal, ou 1FN, é necessário que ela:

- Não possua colunas multivaloradas e/ou grupos repetitivos;
- Todas as suas colunas devem estar no formato atômico, ou seja, não devem ser compostas por múltiplas partes;
- Possua uma chave-primária única, que identifique apenas um registro na tabela.

5.1.2. 2FN

A segunda forma normal lida com tabelas que possuem chaves-primárias compostas. Nesse contexto, para estar em conformidade com a 2FN, uma tabela deve:

- Estar em conformidade com a 1FN;
- Não conter colunas que possuam dependência parcial da chave primária (ou seja, cada coluna da tabela deve depender totalmente da chave primária composta e não de parte dela apenas).

5.1.3. 3FN

Por último, para ser considerada aderente à 3FN, uma tabela deve:

- Estar em conformidade com a 2FN;
- Não possuir dependência transitiva entre as colunas que não pertençam à chave-primária;
- Todas as colunas, que não compõem a chave-primária, devem depender exclusivamente dela.

Como mencionado, essas são as principais formas normais, utilizadas na normalização de uma tabela em bancos de dados relacionais. Entretanto, há outras formas normais, não abordadas aqui, como a Forma Normal Boyce-Codd (FNBC), a 4FN e a 5FN, por exemplo.

6. Linguagem SQL

A implementação física de um banco de dados, por mais que possa ser realizada visualmente, através de softwares específicas, consiste na execução de uma série de comandos (até mesmo por esses softwares mencionados), pertencentes a uma linguagem específica, chamada SQL - sigla que significa Structured Query Language, ou Linguagem Estruturada para Consulta. A SQL é, ao mesmo tempo, uma linguagem padronizada, que segue padrões mantidos pelo ANSI/ISO, como uma linguagem proprietária, já que os fornecedores de softwares SGBD acabam criando variações da mesma. Além disso, essa linguagem é dividida/organizada em subgrupos, de acordo com os tipos de instruções disponíveis:

- DDL: Data Definition Language. Aqui são descritos os comandos destinados à manipulação das estruturas - tabelas, colunas e relacionamentos, definição e descrição dos dados. Logo, a implementação física de um banco de dados faz uso desse tipo de comando, composto por instruções como CREATE|ALTER|DROP TABLE, por exemplo.
- DML: Data Manipulation Language. Nesse conjunto encontram-se as instruções para manipulação dos dados em si, como os comandos INSERT|UPDATE|DELETE;
- DQL: Data Query Language. Composta basicamente pela instrução SELECT, acompanhada de comandos ou cláusulas adicionais.

Além da divisão acima, é comum encontrarmos outros subconjuntos de instruções, como o DCL (Data Control Language), por exemplo, além de uma organização um pouco diferente dessa vista acima. Entretanto, mais importante que a divisão ou organização em si, é entender a sintaxe de cada comando. Como já mencionado, ao implementarem e manterem a compatibilidade com o padrão SQL, os SGBDs permitem que uma mesma

instrução seja executada em diferentes ambientes. Por outro lado, cada SGBD pode também implementar sintaxes distintas, que fogem ao padrão em questão. Logo, é sempre recomendado ler a documentação oficial, para acesso completo aos comandos e funcionalidades disponíveis.

6.1. DDL

Abaixo são demonstrados alguns exemplos de instruções do tipo DDL:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric
);
```

Essa primeira instrução cria uma nova tabela, chamada ‘products’, composta pelas colunas name e price. Além disso, para cada coluna, define o seu tipo de dado - text e numeric, respectivamente. A sintaxe consiste em utilizar as palavras reservadas CREATE TABLE seguidas pelo nome da tabela e, dentro de parênteses, cada coluna que comporá a tabela. Além do tipo de dados, outras informações podem ser adicionadas a cada coluna, como a definição da PRIMARY KEY, a definição de restrições, como NOT NULL, entre outras.

```
DROP TABLE products;
```

Essa segunda instrução remove uma tabela e sua sintaxe é composta pelas palavras reservadas DROP TABLE seguidas do nome da tabela.

```
ALTER TABLE products ADD COLUMN description text;
```

Por fim, temos a instrução de alteração da tabela. Aqui há várias possibilidades, como a exibida acima, onde uma nova coluna é adicionada a uma tabela anteriormente criada, além de outras como a alteração do tipo de dado ou de alguma restrição de uma coluna, a remoção da coluna em si, entre outras.

6.2. DML

```
INSERT INTO products VALUES (1, 'Cheese', 9.99);

INSERT INTO products (product_no, name, price) VALUES
    (1, 'Cheese', 9.99),
    (2, 'Bread', 1.99),
    (3, 'Milk', 2.99);
```

Esses dois primeiros comandos são do tipo INSERT, utilizados, como o nome indica, na inserção de registros em uma tabela. Tal inserção pode ser de um ou mais conjunto de dados, como exibido no segundo exemplo. Além disso, a sintaxe acima deve ser respeitada, onde inicialmente utilizamos as palavras reservas INSERT INTO, seguidas pelo nome da tabela, pelos nomes das colunas (opcional - repare entre os dois exemplos) e pelos valores em si.

UPDATE products SET price = 10 WHERE price = 5;

Essa segunda instrução é do tipo UPDATE e atualiza os dados em uma tabela seguindo um determinado critério. Sua sintaxe é composta pelo comando UPDATE, seguido do nome da tabela e, a seguir, pelo comando SET e pela(s) coluna(s) a ser alterada com seu(s) respectivo(s) novo(s) valor(es). Por fim, e muito importante, temos a cláusula WHERE, responsável por informar qual registro específico deverá ser alterado. Tenha em mente que a omissão da cláusula WHERE pode fazer com que , erroneamente, todos os dados de uma tabela sejam alterados, quando, na verdade, a intenção era modificar apenas um ou um conjunto de dados.

DELETE FROM products WHERE price = 10;
DELETE FROM products;

Nessas duas últimas instruções são demonstradas as instruções DELETE. No primeiro caso estamos deletando o ou os registros da tabela products cuja coluna price tenha o valor 10. Na segunda instrução, estamos deletando todos os registros da tabela products, sem estabelecer nenhum critério. Com isso, podemos ver aqui a sintaxe de tal instrução, composta pelas palavras reservadas DELETE FROM, seguidas do nome da tabela e, por fim, da ou das condições, quando for o caso. Da mesma forma que mencionado ao tratarmos o UPDATE, a utilização do DELETE merece atenção para evitar a remoção indevida de registros.

6.3. DQL

```
SELECT select_list FROM table_expression
```

A instrução acima apresenta a versão mais simples do comando SELECT. Em sua sintaxe temos o comando em si, seguido de uma lista de colunas ou de uma única coluna ou até mesmo de todas as colunas (que pode ser abreviado com o caracter especial *). A seguir temos a palavra reservada FROM, seguida pelo nome da tabela. Esse comando possui uma série de possíveis complementos, sejam eles provenientes da cláusula WHERE, sejam eles de outras instruções, como Group By, Order By, entre outras. A lista de possibilidades pode ser vista a seguir:

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
* | expression [ AS output_name ] [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY expression [, ...] ]
[ HAVING condition [, ...] ]
[ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
[ ORDER BY expression [ ASC | DESC | USING operator ] [, ...] ]
[ LIMIT { count | ALL } ]
[ OFFSET start ]
[ FOR UPDATE [ OF table_name [, ...] ] ]
```

6.3.1. Group By e Having

A cláusula Group By agrupa os dados retornados por uma consulta SELECT com base em determinados grupos, definidos com a utilização da instrução em questão. Tal cláusula normalmente é utilizada em conjunto com funções de agregação, como SUM e COUNT, por exemplo. Já a instrução Having é aplicada sobre o resultado do agrupamento, podendo limitar ou condicionar seu resultado.

A sintaxe de ambas pode ser vista abaixo:

```
SELECT
  column_1,
  column_2,
  ...,
  aggregate_function(column_3)
FROM
  table_name
GROUP BY
```

```
  column_1,
  column_2,
  ...
HAVING
        aggregate_function(column_3) >= 10
```

6.3.2. Order By

A cláusula Order By, como seu nome sugere, aplica ordenação sobre o resultado de uma instrução SELECT. Tal ordenação, por default, é crescente, podendo ser decrescente com a utilização do parâmetro adicional DESC. Além disso, é possível ordenar o resultado de uma consulta por mais de uma coluna. Nesse caso a ordenação seguirá a ordem em que as colunas forem informadas, após a instrução em questão.

6.3.3. Limit e Offset

Essas cláusulas influenciam na quantidade de registros retornada por uma consulta. Com a Limit podemos limitar a quantidade em si e com a Offset definirmos a partir de qual registro queremos retornar os dados (ou, em outras palavras, quais as linhas deverão ser ignoradas no retorno). Ambas são muito utilizadas para a paginação de grandes quantidades de resultados. Veja a sintaxe de utilização:

```
SELECT
  column_1,
  column_2,
  ...,
FROM
  table_name
LIMIT 10
OFFSET 5
```

A consulta acima retornará 10 registros (LIMIT 10) pulando os 5 primeiros registros (OFFSET 5).

6.4. Junção

Embora pertencentes ao conjunto de instruções DQL, as funções de junção merecem um capítulo à parte. Em linhas gerais, essas funções são utilizadas para a recuperação de dados de duas ou mais tabelas. Para isso, as tabelas precisam de uma coluna que as relacione (normalmente o par chave-primária x chave-estrangeira). Além disso, há diferentes formas de junção, baseadas também na teoria dos conjuntos, na disciplina de matemática. As principais funções de junção são INNER JOIN, LEFT OUTER JOIN E RIGHT OUTER JOIN.

6.4.1. INNER JOIN

Essa junção retorna o conjunto de dados correspondente à intersecção entre duas (ou mais) tabelas. Veja as tabelas-exemplo abaixo:

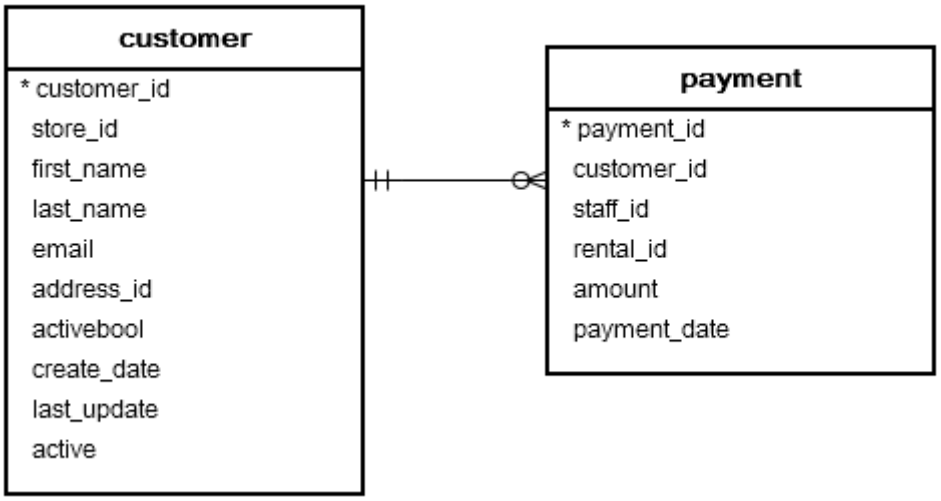

Figura 3: Tabelas customer e payment (Fonte: postgresqltutorial.com, 2022)

A relação acima implica que, para cada pagamento inserido na tabela payment, será informado o customer correspondente, através da coluna customer_id. Logo, essa é a coluna de ligação entre ambas as tabelas. A query abaixo retornará os dados de customer e payment para todos os registros de payment cujo customer_id exista na tabela customer - em outras palavras: para cada linha da tabela B (payment) deverá existir um correspondente (customer_id) na tabela A (customer).

```
SELECT
        c.first_name,
        c.last_name,
        p.amount,
        p.payment_date
```

```
FROM
        customer c
INNER JOIN payment p
   ON c.customer_id = p.customer_id
```

6.4.2. LEFT OUTER JOIN

As junções do tipo OUTER retornam os dados de uma junção, independente de haver correspondência entre cada elemento das tabelas relacionadas. Utilizando as tabelas acima, imagine que um exista um customer inserido na tabela em questão e que não tenha feito nenhum pagamento - consequentemente, não possua nenhum registro com seu identificador (customer_id) na tabela payment. Nesse caso, a consulta vista anteriormente, INNER JOIN, não traria nenhuma informação do customer (por não haver intersecção entre as tabelas, conforme já explicado). Já a consulta abaixo retornaria os dados do customer e nas linhas correspondentes ao payment seria retornado NULL - uma vez que não há linhas nessa tabela referentes ao customer utilizado aqui como exemplo.

```
SELECT
        c.first_name,
        c.last_name,
        p.amount,
        p.payment_date
FROM
        customer c
LEFT JOIN payment p
   ON c.customer_id = p.customer_id
```

Cabe destacar que o LEFT ou RIGHT, nas instruções OUTER JOIN, dizem respeito à tabela posicionada à esquerda ou à direita na cláusula ON (nesse caso, customer).

6.4.3. RIGHT OUTER JOIN

Por fim, o RIGHT JOIN é similar ao LEFT, sendo que, aqui, são considerados os registros da tabela à direita. Logo, e mantendo o mesmo exemplo acima, onde temos as tabelas customer e payment, a instrução abaixo retornaria os dados da tabela payment, independente de haver um cliente correspondente (por exemplo, caso um pagamento tenha sido lançado e mantido para um customer que não exista mais na tabela em questão).

```
SELECT
        c.first_name,
```

```
        c.last_name,
        p.amount,
        p.payment_date
FROM
        customer c
RIGHT JOIN payment p
   ON c.customer_id = p.customer_id
```

Referências Bibliográficas

PUGA, Sandra. FRANÇA, Edson. GOYA, Milton. Banco de dados: Implementação em SQL, PL/SQL e Oracle 11g. Editora Pearson, 2013.

POSTGRESQL TUTORIAL. PostgreSQL INNER JOIN. Disponível em: <https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-inner-join//>. Acesso em: 05 de set de 2022.