Projeto Físico

- O projeto físico do banco de dados consiste no mapeamento do projeto lógico para um DBMS real
- Projeto deve levar em conta fatores como:
  - Desempenho
  - Tempo de resposta das transações
  - Alocação de espaço em disco
- Principais definições
  - Escolher SGBD
  - Definir estrutura do banco de dados (Schema)
  - Definir índices
  - Definir tamanhos de bloco (páginas)
  - Definir localização física de arquivos e índices
  - Definir permissões de acesso (grupos, usuários e papéis)

Projeto Físico

- Principais critérios
  - Espaço disponível
  - Freqüência de execução de consultas
  - Freqüência de transações de atualização do BD
  - Pico de transações correntes
  - Restrições de integridade de atributos

Projeto Físico
Definição da Estrutura do BD

- A definição é baseada da linguagem SQL

ANO	NOME	APELIDO	COMENTÁRIOS
1974			System R. Proposto pela IBM (Raymond F. Boyces) baseado no modelo relacional proposto por Edgar F. Codd. Origem da linguagem SQL
1986	SQL-86	SQL-87	Publicado pela ANSI e adotado como padrão em 1987
1989	SQL-89	FIPS 127-1	Revisão. Adotado como FIPS 127-1 (Federal Information Processing Standards Publications)
1992	SQL-92	SQL2, FIPS 127-2	Segunda revisão. Adotada como FIPS 127-2 e também pela ISO (ISO/IEC 9075:1992)
1999	SQL:1999	SQL3	Novos recursos: Expressões regulares, consultas recursivas, triggers, suporte a procedimentos e controle de fluxo alguns conceitos de orientação a objetos
2003	SQL:2003		Introdução de recursos para manipulação de XML; Colunas com auto-incremento e identidades
2006	SQL:2006		ISO/IEC 9075-14:2006 define formas que a SQL pode ser utilizada em conjunto com XML como importar e armazenar um XML. Integração do código SQL com Xquery
2008	SQL:2008		Ultima revisão

Linguagem SQL

- Linguagem SQL pode ser dividida alguns grupos de funcionalidades
- Dois são deles são básicos para criação e manipulação de banco de dados
  - SQL DDL (Data Definition Language)
    - Permite a definição da estrutura (schema) do banco de dados
    - CREATE, ALTER, DROP
      - Database, Schema, Table, Index etc.
  - SQL DML (Data Manipulation Language)
    - Permite a manipulação de dados
    - INSERT, SELECT, UPDATE, DELETE

Linguagem SQL
DDL – Create Database

- Criação de um database
- Sintaxe:
  - CREATE DATABASE name
    [[WITH] [OWNER [=] dbowner]
    [TEMPLATE [=] template]
    [ENCODING [=] encoding]
    [TABLESPACE [=] tablespace]
    [CONNECTION LIMIT [=] connlimit]
- Exemplos:
  - CREATE DATABASE dbRH;
  - CREATE DATABASE "dbRH" WITH ENCODING='UTF8' OWNER=root;

Linguagem SQL
DDL – Create Schema

- Criação de um schema
- Sintaxe:
  - CREATE SCHEMA schemaname
    [AUTHORIZATION username] [schema_element [...]]
- Exemplos:
  - CREATE SCHEMA myschema;
  - CREATE SCHEMA hollywood
    CREATE TABLE films (title text, release date, awards text[])
    CREATE VIEW winners AS
    SELECT title, release FROM films WHERE awards IS NOT NULL;

Linguagem SQL
DDL – Create Table

- Permite a definição das tabelas
- Tabela pertence ao usuário que a criou
- Sintaxe:

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name
(
 [
  {
      column_name data_type [ DEFAULT default_expr ] [ column_constraint [ ... ] ]
      | table_constraint
      | LIKE parent_table [ { INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES } ]...
  }
  [, ... ]
 ]
)
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
```

Linguagem SQL
DDL – Create Table - Column Constraints

- Restrições aplicadas em colunas

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  CHECK ( expression ) |
  REFERENCES reftable [ ( refcolumn ) ]
                              [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
                               [ON DELETE action ]
                               [ON UPDATE action]
}
[DEFERRABLE | NOT DEFERRABLE]
[INITIALLY DEFERRED | INITIALLY IMMEDIATE]
```

- index_parameters utilizado nas restrições UNIQUE e PRIMARY KEY

```
    [ WITH ( storage_parameter [= value] [, ... ] ) ]
    [ USING INDEX TABLESPACE tablespace ]
```

Linguagem SQL
DDL – Create Table - Table Constraints

- Restrições aplicadas na tabela

```
[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  CHECK ( expression ) |
  FOREIGN KEY ( column_name [, ... ] )
    REFERENCES reftable [( refcolumn [, ... ] )]
                        [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
                        [ ON DELETE action ]
                        [ ON UPDATE action ]
}
[ DEFERRABLE | NOT DEFERRABLE ]
[ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- index_parameters utilizado nas restrições UNIQUE e PRIMARY KEY

```
[ WITH ( storage_parameter [= value] [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace ]
```

Linguagem SQL
DDL – Create Table - Exemplos

- Exemplos:

```
CREATE TABLE films
(
   code char(5) CONSTRAINT firstkey PRIMARY KEY,
   title varchar(40) NOT NULL,
   did integer NOT NULL,
   date_prod date,
   kind varchar(10),
   len interval hour to minute
);

CREATE TABLE distributors
(
   did integer PRIMARY KEY DEFAULT nextval('serial'),
   name varchar(40) NOT NULL CHECK (name <> '')
);
```

Linguagem SQL
DDL – Create Table - Exemplos

- Exemplos:
  - Define uma constraint para a tabela, que pode ser aplicada a mais de uma coluna

```
CREATE TABLE films
(
   code char(5),
   title varchar(40),
   did integer,
   date_prod date,
   kind varchar(10),
   len interval hour to minute,
   CONSTRAINT production UNIQUE(date_prod)
);
```

Linguagem SQL
DDL – Create Table - Exemplos

- Exemplos:
  - Constraint de Coluna versus constraint de tabela:

```
CREATE TABLE distributors
(
   did integer CHECK (did > 100),
   name varchar(40) CHECK (name <> ”)
);
```

  - Mesma constraint acima, porém neste caso associada à tabela:

```
CREATE TABLE distributors
(
   did integer,
   name varchar(40),
   CONSTRAINT con1 CHECK (did > 100 AND name <> ”)
);
```

Linguagem SQL
DDL – Create Table - Exemplos

- Exemplos (Definição da Chave primária)
  - Definição feita como uma CONSTRAINT

```
CREATE TABLE films  (
   code char(5),
   title varchar(40),
   did integer,
   date_prod date,
   kind varchar(10),
   len interval hour to minute,
   CONSTRAINT code_title PRIMARY KEY(code,title)
);
```

  - Definição feita como uma CONSTRAINT associada à tabela e em seguida associada à coluna

```
CREATE TABLE distributors (
   did integer,
   name varchar(40),
   PRIMARY KEY(did)
);
CREATE TABLE distributors (
   did integer PRIMARY KEY,
   name varchar(40)
);
```

Linguagem SQL
DDL – Create Table - Exemplos

- Exemplos
  - Associada um valor default a uma coluna
  - Coluna **did** contem um valor default que equivale ao próximo valor disponível para a sequence “distributors_serial”
  - Coluna **modtime** contém como valor default a data e hora em que a linha foi inserida

```
CREATE SEQUENCE distributors_serial;

CREATE TABLE distributors
(
  name varchar(40) DEFAULT 'Luso Films',
  did integer DEFAULT nextval('distributors_serial'),
  modtime timestamp DEFAULT current_timestamp
);
```

Linguagem SQL – DDL
Create Table – Chave Estrangeira

- Exemplos

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

- Utilizando uma constraint para a tabela

```
CREATE TABLE t1 (
    a integer PRIMARY KEY,
    b integer,
    c integer,
    FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

Linguagem SQL – DDL
Create Table – Chave Estrangeira

- Exemplos

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

- Utilizando uma constraint para a tabela

```
CREATE TABLE t1 (
    a integer PRIMARY KEY,
    b integer,
    c integer,
    FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

Linguagem SQL – DDL
Create Table – Chave Estrangeira

- Exemplos – Cascade / Restrict

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    shipping_address text,
    -- ...
);
CREATE TABLE order_items (
    product_no integer REFERENCES products ON DELETE RESTRICT,
    order_id integer REFERENCES orders ON DELETE CASCADE,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

Linguagem SQL
DDL – Create Index

- Criação de índices
- Sintaxe

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] name ON table [ USING method ]
(
    { column | ( expression ) }[ opclass ][ ASC | DESC ][ NULLS { FIRST | LAST } ][, ...]
)
[ WITH ( storage_parameter = value [, ... ] ) ]
[ TABLESPACE tablespace ]
[ WHERE predicate ]
```

- Exemplos
  - CREATE UNIQUE INDEX title_idx ON films (title);
  - CREATE INDEX lower_title_idx ON films ((lower(title)));
  - CREATE INDEX title_idx_nulls_low ON films (title NULLS FIRST);

Linguagem SQL
DDL – Alter Database e Schema

- Modifica parâmetros de um database já criado
  - ALTER DATABASE name [[WITH] option [...]]
    - Neste caso option pode assumir o valor:

    CONNECTION LIMIT connlimit
  - ALTER DATABASE name RENAME TO newname
  - ALTER DATABASE name OWNER TO new_owner
  - ALTER DATABASE name SET configuration_parameter { TO | = } { value | DEFAULT }
  - ALTER DATABASE name SET configuration_parameter FROM CURRENT
  - ALTER DATABASE name RESET configuration_parameter
  - ALTER DATABASE name RESET ALL
- Modifica parâmetros de um schema já criado
  - ALTER SCHEMA name RENAME TO newname
  - ALTER SCHEMA name OWNER TO newowner

Linguagem SQL
DDL – Alter Table

- Modifica uma tabela
- Muito comum seu uso a fim de permitir a criação de referências entre as tabelas
  - ALTER TABLE [ONLY] name [*]
    action [, ...]
  - ALTER TABLE [ONLY] name [*]
    RENAME [COLUMN] column TO new_column
  - ALTER TABLE name
    RENAME TO new_name
  - ALTER TABLE name
    SET SCHEMA new_schema

Linguagem SQL
DDL – Alter Table

- Onde action pode conter o valor
  - ADD [COLUMN] column type [column_constraint [...]]
  - DROP [COLUMN] column [RESTRICT | CASCADE]
  - ALTER [COLUMN] column TYPE type [USING expression]
  - ALTER [COLUMN] column SET DEFAULT expression
  - ALTER [COLUMN] column DROP DEFAULT
  - ALTER [COLUMN] column { SET | DROP } NOT NULL
  - ALTER [COLUMN] column SET STATISTICS integer
  - ALTER [COLUMN] column SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
  - ADD table_constraint
  - DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]

Linguagem SQL
DDL – Alter Table

- Onde action pode conter o valor
  - DISABLE TRIGGER [trigger_name | ALL | USER]
  - ENABLE TRIGGER [trigger_name | ALL | USER]
  - ENABLE REPLICA TRIGGER trigger_name
  - ENABLE ALWAYS TRIGGER trigger_name
  - DISABLE RULE rewrite_rule_name
  - ENABLE RULE rewrite_rule_name
  - ENABLE REPLICA RULE rewrite_rule_name
  - ENABLE ALWAYS RULE rewrite_rule_name
  - CLUSTER ON index_name
  - SET WITHOUT CLUSTER
  - SET WITHOUT OIDS
  - SET (storage_parameter = value [, ...])
  - RESET (storage_parameter [, ...])
  - INHERIT parent_table
  - NO INHERIT parent_table
  - OWNER TO new_owner
  - SET TABLESPACE new_tablespace

Linguagem SQL
DDL – Alter Table - Exemplos

- Adicionar uma coluna:
  - ALTER TABLE distributors ADD COLUMN address varchar(30);
- Remover uma coluna:
  - ALTER TABLE distributors DROP COLUMN address RESTRICT;
- Alterar o tipo de dados de colunas:
  - ALTER TABLE distributors
    ALTER COLUMN address TYPE varchar(80),
    ALTER COLUMN name TYPE varchar(100);
- Renomear uma coluna:
  - ALTER TABLE distributors RENAME COLUMN address TO city;
- Renomear uma tabela:
  - ALTER TABLE distributors RENAME TO suppliers;
- Adicionar uma constraint na coluna:
  - ALTER TABLE distributors ALTER COLUMN street SET NOT NULL;
- Remover uma constraint de uma coluna:
  - ALTER TABLE distributors ALTER COLUMN street DROP NOT NULL;

Linguagem SQL
DDL – Alter Table - Exemplos

- Adicionar uma constraint the check à tabela:
  - ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK (char_length(zipcode) = 5);
- Remover uma constraint de uma tabela e todos seus filhos:
  - ALTER TABLE distributors DROP CONSTRAINT zipchk;
- Adicionar uma chave estrangeira à tabela
  - ALTER TABLE distributors ADD CONSTRAINT distfk FOREIGN KEY (address) REFERENCES addresses (address) MATCH FULL;
  - ALTER TABLE products ADD FOREIGN KEY (product_group_id) REFERENCES product_groups;
  - ALTER TABLE products ADD COLUMN product_group_id integer REFERENCES product_groups;
- Adicionar uma constraint UNIQUE a mais de uma coluna da tabela:
  - ALTER TABLE distributors ADD CONSTRAINT dist_id_zipcode_key UNIQUE (dist_id, zipcode);
- Adicionar uma chave primária à tabela
  - ALTER TABLE distributors ADD PRIMARY KEY (dist_id);
- Alterar o tablespace da tabela
  - ALTER TABLE distributors SET TABLESPACE fasttablespace;
- Mover a tabela para outro schema
  - ALTER TABLE myschema.distributors SET SCHEMA yourschema;

Linguagem SQL
DDL – Alter Index

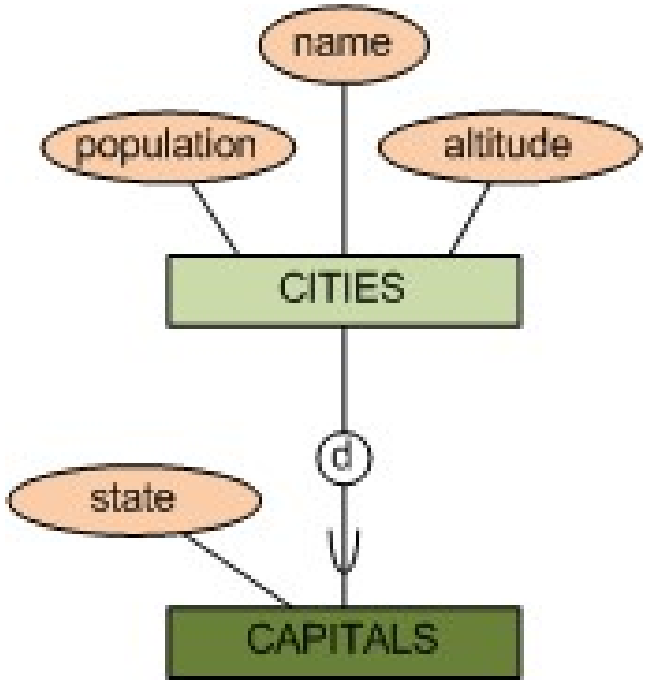
- Modifica um índice
  - ALTER INDEX name RENAME TO new_name
  - ALTER INDEX name SET TABLESPACE tablespace_name
  - ALTER INDEX name SET (storage_parameter = value [, ...])
  - ALTER INDEX name RESET (storage_parameter [, ...])
- Exemplos
  - Renomear um índice
    - ALTER INDEX distributors RENAME TO suppliers;
  - Mover para um outro tablespace
    - ALTER INDEX distributors SET TABLESPACE fasttablespace;

Linguagem SQL
DDL – Drop

- Database
  - DROP DATABASE [IF EXISTS] name
- Schema
  - DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
  - Exemplo
    - DROP SCHEMA mystuff CASCADE;
- Table
  - DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
    - Para remover apenas o conteúdo de uma tabela pode ser utilizado os comandos DELETE ou TRUNCATE
  - Exemplo
    - DROP TABLE films, distributors;
- Index
  - DROP INDEX [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
  - Exemplo
    - DROP INDEX title_idx;

Linguagem SQL
DDL - Herança

- A partir do SQL 99 a linguagem possui recursos para acomodar conceitos de orientação a objetos
- Considerando o EER abaixo



- A herança pode ser representada no modelo relacional da seguinte forma:

Linguagem SQL
DDL - Herança

- Representação Modelo Relacional

```
CREATE TABLE cities (
    name text,
    population float,
    altitude int
);

CREATE TABLE capitals (
    state char(2)
) INHERITS (cities);
```

Linguagem SQL – DDL
Herança – Manipulação dos Dados

- Manipulação dos dados
  - Seleção de todas as cidades, incluindo capitais

```
SELECT name, altitude
FROM cities
WHERE altitude > 500;
```

  - Seleção somente das cidades, excluindo as capitais

```
SELECT name, altitude
FROM ONLY cities
WHERE altitude > 500;
```

  - Obtenção do nome da tabela

```
SELECT c.tableoid, c.name, c.altitude
FROM cities c
WHERE c.altitude > 500;
```

Linguagem SQL – DDL
Herança – Manipulação dos Dados

- Comando INSERT não propaga a herança para outras tabelas
- O comando abaixo falha

```
INSERT INTO cities (name, population, altitude, state)
VALUES ('New York', NULL, NULL, 'NY')
```

- Inserção ocorre apenas na tabela especificada

```
INSERT INTO cities (name, population, altitude)
VALUES ('New York', NULL, NULL)
INSERT INTO capitals (state)
VALUES ('NY')
```

Linguagem SQL – DDL
Herança – Manipulação dos Dados

- Manipulação dos dados – Obtenção do Nome da Tabela
  - tableoid – Campo presente em qualquer tabela definida no POSTGRES
  - Indica a tabela

```
SELECT c.tableoid, c.name, c.altitude
FROM cities c
WHERE c.altitude > 500;
```

  - PG_CLASS – Tabela que contém definições das tabela de um schema

```
SELECT p.relname, c.name, c.altitude
FROM cities c, pg_class p
WHERE c.altitude > 500
and c.tableoid = p.oid;
```

Linguagem SQL
DML – Data Manipulation Language

- A SQL/DML(Data Manipulation Language) é um subconjunto da SQL usada para selecionar, inserir, atualizar e remover dados de tabelas do banco de dados
- Principais comandos
  - INSERT
  - DELETE
  - UPDATE
  - SELECT

Linguagem SQL
DML – INSERT

- Insere uma linha em uma tabela
- Sintaxe

```
INSERT INTO table [ ( column [, ...] ) ]
{ DEFAULT VALUES |
  VALUES ( { expression | DEFAULT } [, ...] ) [, ...] |
  query
}
```

- Exemplos
  - INSERT INTO films (code, title, did, date_prod, kind) VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
  - INSERT INTO films VALUES ('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82 minutes');
  - INSERT INTO films SELECT * FROM tmp_films WHERE date_prod < '2004-05-07';

Linguagem SQL
DML – INSERT - Exemplos

- Outros Exemplos
  - INSERT INTO employee

    VALUES ('John', 'B', 'Smith', '123456789',DATE '1965-01-09', '731 Fondren, Houston, TX','M', 30000, '333445555', 5);

  - INSERT INTO employee(fname, minit, lname, ssn)

    VALUES ('John', 'B', 'Smith', '123456789');

  - INSERT INTO works_onSELECT ssn, pnumber, 0FROM employee, project;

Linguagem SQL
DML – DELETE

- Remove linhas de uma tabela
- Sintaxe
  - DELETE FROM [ONLY] *table* [[AS] *alias*] [USING *usinglist*]
    [WHERE *condition* | WHERE CURRENT OF *cursor_name*]
- Exemplos
  - Remove todas as linhas da tabela
    - DELETE FROM films;
  - Remove apenas linhas que atendem ao critério
    - DELETE FROM films WHERE kind <> 'Musical';
  - Remove apenas a linha onde o cursor está posicionado
    - DELETE FROM tasks WHERE CURRENT OF c_tasks;

Linguagem SQL
DML – UPDATE

- Altera o conteúdo de linhas de uma tabela
- Sintaxe

```
UPDATE [ ONLY ] table [ [ AS ] alias ]
SET {  column = { expression | DEFAULT } |
          ( column [, ...] ) = ( { expression | DEFAULT } [, ...] )
        } [, ...]
[ FROM fromlist ]
[ WHERE condition | WHERE CURRENT OF cursor_name ]
[ RETURNING * | output_expression [ AS output_name ] [, ...] ]
```

Linguagem SQL
DML – UPDATE - Exemplos

- Exemplos
  - UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
  - UPDATE weather SET temp_lo = temp_lo+1, temp_hi = temp_lo+15, prcp = DEFAULT WHERE city = 'San Francisco' AND date = '2003-07-03';
  - UPDATE weather SET temp_lo = temp_lo+1, temp_hi = temp_lo+15, prcp = DEFAULT WHERE city = 'San Francisco' AND date = '2003-07-03' RETURNING temp_lo, temp_hi, prcp;
  - UPDATE employees SET sales_count = sales_count + 1 FROM accounts WHERE accounts.name = 'Acme Corporation' AND employees.id = accounts.sales_person;
  - UPDATE employees SET sales_count = sales_count + 1 WHERE id = (SELECT sales_person FROM accounts WHERE name = 'Acme Corporation');

Linguagem SQL
DML – SELECT

- Retorna as linhas de uma tabela
- Sintaxe

```
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 ] [ NULLS { FIRST |
    LAST } ] [, ...] ]
[ LIMIT { count | ALL } ]
[ OFFSET start ]
[ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [...] ]
```

Linguagem SQL
DML – SELECT

- O termo from_item pode ser da seguinte forma:
  - [ONLY] table_name [*] [[AS] alias [(column_alias [, ...])]]
  - (select) [AS] alias [(column_alias [, ...])]
  - function_name ([argument [, ...]]) [AS] alias [(column_alias [, ...] | column_definition [, ...])]
  - function_name ([argument [, ...]]) AS (column_definition [, ...])
  - from_item [NATURAL] join_type from_item [ON join_condition | USING (join_column [, ...])]

Transações

- Transação é uma unidade de trabalho em um programa executando;
  - ACID (Atomicidade, Consistência, Isolamento e Durabilidade)
  - Atomicidade
    - Ou se realiza por completo ou é cancelada
  - Consistência
    - O banco de dados deve ficar íntegro
  - Isolamento
    - Transações são isoladas uma das outras
    - Várias instâncias de uma transação não podem colidir
  - Durabilidade
    - Assim que é concluída seu efeitos são mantidos, ainda que o sistema fique indisponível

Transações

- Iniciar a transação
  - BEGIN [WORK | TRANSACTION]
  - No SQL92 o inicio da transação é implícito
- Confirmar
  - COMMIT [WORK | TRANSACTION]
  - Padrão: AUTOCOMMIT On (cada comando é uma transação)
- Cancelar
  - ROLLBACK [WORK | TRANSACTION]

Concorrência

- OBJETIVO
  - Garantir a disponibilidade e integridade, evitando anomalias
- Leitura de Sujeira (dirty read)
  - Leitura de dados atualizados e não efetivado (uncommited) por outra transação
- Leitura não repetível
  - Leitura de dados, seguida por de atualização destes mesmos dados por outra transação antes a atual termine
- Fantasma
  - Leitura de um conjunto de linhas, seguido de inserção de linhas por outra transação antes que a atual termine
- Postgres
  - Implementa o Multi Version Concurrency Control (MVCC)

Postrgres - MVCC

- Cada transação possui uma visão do BD
- Trancas de leitura não conflitam com trancas de gravação
- Permite trancas explícitas em tabelas ou linhas

Níveis de Isolamento

- Serializable
  - Cada transação enxerga o estado do BD efetivado antes da sua primeira consulta
- Read Committed
  - Uma consulta enxerga do estado do BD efetivado antes de sua execução sendo possível ocorrer Fantasmas e Leituras não repetíveis
- PostgreSQL não implementa
  - Read Uncommitted - Permite as três anomalias citadas
  - Repeatable Read – Permite apenas fantasmas

Níves de Isolamento

- Níveis de Isolamento x Anomalias

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

Implementação dos Níveis de Isolamento

SET TRANSACTION ISOLATION LEVEL
{ READ COMMITTED | SERIALIZABLE }
SET SESSION CHARACTERISTICS AS TRANSACTION · · ·

- SET TRANSACTION sobrepõe SET SESSION
- SET default transaction isolation = ’value’

Trancas (Locks)

- Controle Automático
  - ALTER TABLE - Tranca a tabela para qualquer outra operação
  - DELETE ou UPDATE - Tranca linha para outras gravações
  - SELECT FOR UPDATE - Tranca linha para gravações
- Trancas Explícitas em Tabelas
  - LOCK [TABLE] name [, ...]
  - LOCK [TABLE] name [, ...] IN lockmode MODE
  - lockmode: {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE }

Trancas (Locks) - LockMode

- Trancas Explícitas em Tabelas
  - LOCK TABLE name [, ...]
  - LOCK TABLE name [, ...] IN lockmode MODE
    - lockmode: {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE }
  - ACCESS SHARE conflita com: access exclusive;
  - ROW SHARE conflita com: exclusive e access exclusive;
  - ROW EXCLUSIVE conflita com: share, share row exclusive, exclusive e access exclusive;
  - SHARE UPDATE EXCLUSIVE conflita com: share update exclusive, share, share row exclusive, exclusive e access exclusive;
  - SHARE conflita com: row exclusive, share update exclusive, share row exclusive, exclusive e access exclusive;
  - SHARE ROW EXCLUSIVE conflita com: row exclusive, share update exclusive, share, share row exclusive, exclusive e access exclusive;
  - EXCLUSIVE conflita com: share, row exclusive, share update exclusive, share row exclusive, exclusive e access exclusive;
  - ACCESS EXCLUSIVE: conflita com todos os modos, garantindo que somente um usuário está acessando a tabela (DEFAULT)

Deadlocks

- O Postgresql automaticamente detecta o deadlock e cancela uma das transações, por exemplo.
  - T1: adquire tranca exclusiva para Tabela A
  - T1: tenta obter uma tranca para uma Tabela B, mas
  - T2: já tem uma tranca exclusiva para Tabela B
  - T1: aguarda liberação da tranca da Tabela B
  - T2: tenta obter uma tranca para Tabela A
- CONSISTÊNCIA
  - O Postgresql por default não tranca dados para leitura
- SAVEPOINTS
  - Ver Advance Features/Transaction no manual do PostgreSQL (versão 8.*)