Update：用于改变列值，而非行值。

```
update table_name
set column_name = new_value
    other_col = other_val,
    ...
where ...
```

→ 甚至可以改成 NULL（对非必要列值）

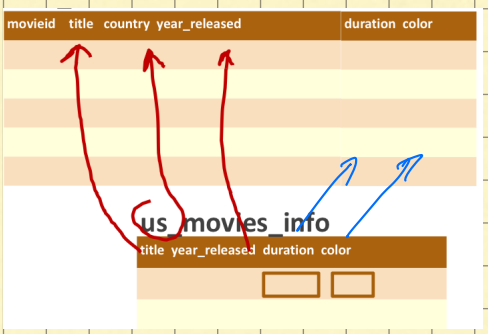
若无 where，所有行都会被影响。

```
update us_movie_info
set title = replace(title, "'", '')
```

```
update people
set surname = substr(surname, 4)
    || ' (von)'
where surname like 'von %'
```

von Strohein → Strohein (von)

update 不能用在 loop 中。




将一个表中的数据 update 到另一个表中时，有点类似于 join 操作（老式 join，即用 where 限制）

```
update movies_2
set (duration, color) =
      (select duration,
          case color
            when 'C' then 'Y'
            when 'B' then 'N'
          end color
       from us_movie_info i
       where i.title = movies_2.title
         and i.year_released = movies_2.year_released)
where country = 'us'
  and exists (select null
              from us_movie_info i2
              where i2.title = movies_2.title
                and i2.year_released = movies_2.year_released)
```

run for each retrieved row

用于判断有无 null 行。

```
update movies_2
set (duration, color) =
      (select duration,
          case color
            when 'C' then 'Y'
            when 'B' then 'N'
          end color
       from us_movie_info i
       where i.title = movies_2.title
         and i.year_released = movies_2.year_released)
where country = 'us'
  and (m.title, m.year_released)
    in (select title, year_released
        from us_movie_info)
```

run for each retrieved row

Once

Oracle and DB2 both support subqueries returning several columns (SQlite also now).

```
update movies_2
set   duration = i.duration,
      color = case i.color
                when 'C' then 'Y'
                when 'B' then 'N'
              end
from us_movie_info i
where i.title = movies_2.title
  and i.year_released = movies_2.year_released
  and movies_2.country = 'us'
```

3 Join

```
update movies_2 m
    inner join us_movie_info i
      on i.title = m.title
     and i.year_released = m.year_released
set m.duration = i.duration,
   m.color = case i.color
               when 'C' then 'Y'
               when 'B' then 'N'
             end
where m.country = 'us'
```

由于 update 会直接改变数据，故要对 join 操作更注意

通常来说，不能修改 key 的值（因为 key 是对该行的标识符），只能删除该行并插入其他行。

Merge：对两个表进行同步操作

```
merge into movies_2 m
using (select 'us' as country,
          title,
          year_released,
          duration,
          case color
            when 'C' then 'Y'
            when 'B' then 'N'
          end as color
       from us_movie_info) i
   on (i.country = m.country
  and i.title = m.title
  and i.year_released = m.year_released)
when matched then
    update
    set m.duration = i.duration,
        m.color = i.color
when not matched then
    insert(title, year_released, country, duration, color)
    values(i.title, i.year_released, i.country, i.duration, i.color)
```

**Update or Insert**

An interesting operation would be to update a film we know, and insert it if we don't. That's the purpose of MERGE.

```
insert into movies_2(title, year_released,
                     country, duration, color)
select title, year_released, country, duration, color
from (select title,
          year_released,
          'us' as country,
          duration,
          case color
            when 'C' then 'Y'
            when 'B' then 'N'
          end color
      from us_movie_info) i
on duplicate key update
   movies_2.duration = i.duration,
   movies_2.color = i.color
```

**Update or Insert**

MySQL can catch an insert that fails because the row is already here, and turn on the fly the insert into an update.

```
insert    or replace    into movies_2(title, year_released,
                                       country, duration, color)
select title, year_released, country, duration, color
from (select title,
          year_released,
          'us' as country,
          duration,
          case color
            when 'C' then 'Y'
            when 'B' then 'N'
          end color
      from us_movie_info) i
```

**Update or Insert**

SQLite allows something similar with a simpler (but less flexible) syntax. Beware, because it deletes a row and creates a new one, foreign keys may not like it.

**Update then Insert**

Update

```
insert into movies_2(title, year_released, country,
                     duration, color)
select i.title, i.year_released, 'us', i.duration,
       case i.color
         when 'C' then 'Y'
         when 'B' then 'N'
       end
from us_movie_info i
   left outer join movies_2 m
     on m.title = i.title
    and m.year_released = i.year_released
    and m.country = 'us'
where m.movieid is null
```

When none of the above is available, you should try to update, and if nothing is affected insert.

# Delete

通常说来，数据不能立刻删除，一般最好是将数据放入历史表中，再从活跃表中删去。

```
delete from table_name
where ...
```

若无where，操作会涉及整个表。

在执行delete之前，会为了rollback保存数据，但会使得操作变慢。
truncate语句（没有where）不能rollback，且操作变快，但不建议用。

# Constraints

= **guarantee**

constraint（尤其是外键）能保证data的一致性。
（在insert、update、delete中）
若有key限制，则不能直接删去。


To delete the row for China in table countries.

The constraint will prevent you to do that.

```
delete from countries where country_code='cn';
```

[23503] ERROR: update or delete on table "countries" violates foreign key constraint "movies_country_fkey" on table "movies"
详细：Key (country_code)=(cn) is still referenced from table "movies".

# Functions（函数）

大部分DBMS都会实现基于sql语言的函数，可直接调用

定义函数：

Here is a PostgreSQL example

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
 return case
        when p_fname is null then ''
        else p_fname || ' '
       end |
        case position('(' in p_sname)
          when 0 then p_sname
          else trim('0' from substr(p_sname,
                    position('(' in p_sname) + 1))
               ||' '
               || trim(substr(p_sname, 1,
                    position('(' in p_sname) - 1))
        end;
end;
$$ language plpgsql;
```

→可指定语言。

使用函数：

```
select full_name(first_name, surname) as name,
    born, died
from people
order by surname
```

**Procedural extensions to SQL**

| | |
|---|---|
| T-SQL | |
| (no name) | |
| PL/SQL | |
| PL/PGSQL | |
| SQL PL | |
| nothing ... | |

You can use C or any language with SQLite. If you crash your program, it only affects you.

Procedural extensions 提供了所有一般编程语言的组件。
同时也支持所有DDL。

**Procedural ?**

- variables
- conditions
- loops
- arrays
- error management

Procedural extensions provide all the bells and whistles of true programming languages (they were often inspired by programming languages such as PL/I or ADA). They are a mixed blessing, because they often incite programmers to do the wrong things with them.

... TRUE PROGRAMMING LANGUAGE

They also support all DML statements (no always DDL, but you can cheat)

```
select col1, col2, ...
into local_var1, local_var2, ...
from ...
```

+ CURSORS

想从DB中获得data到variables中
若查询返回单行，可用 select... into...
若想遍历查询的返回值，可用cursors（实际上是row variables）

row-by-row set processing

有一种查找函数，返回与值相关联的标签。由于它是存在DB中的procedure，许多人都认为这个可以常用。但实际上尽量少用。

```
select country_name(country_code), title, ...
from movies
where ...
```

```
select c.country_name, m.title, ...
from movies m
   inner join countries c
     on c.country_code = m.country
where ...
```

```
create function country_name(p_code varchar2)
return countries.country_name%type
as
  v_name   countries.country_name%type;
begin
  select country_name
  into v_name
  from countries
  where country_code = p_code;
  return v_name;
end;
```

实际上，该函数的功能可用join实现。

而且join有许多方法，其中有一些在大容量数据中特别有效。

这些查找函数强制执行"one row at a time"的join

这是很可怕的，

函数不应该查询DB

如果可能的话，应在一条SQL语句中完成任务。

若不能，则用尽可能少的PL/SQL完成。