SQL

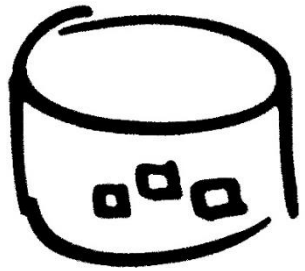
Structured Query Language

Oliver 2018-01-18

OUTLINE

- Structure
- Basic Syntax
- Normalization
- Small Database Design
- Large Database Design

Structure



- Database

Database is a container for us to save Table and other SQL structure.

- Table

Table is a structure to save data which is organized by columns and rows.

- Column/Field

Column is a block of data in Table. Usually, we classify data to be columns.

- Row/Record

Row is a combination of all descriptions of a object.

Basic Syntax

Create Database & Use Database

- Create Database

Syntax:

CREATE DATABASE database_name;

- Use Database

Syntax:

USE database_name;

Create Table

Syntax:

```
CREATE TABLE table_name
(
    column_name1 VARCHAR(10),
    column_name2 VARCHAR(6)
);
```

Insert data

Syntax:

```
INSERT INTO table_name ( column_name1, column_name2, …)
                VALUES(‘value1’, ‘value2’, …);
```

SELECT

- Select all columns

SELECT * FROM table_name;

- Given conditions

E.g.

SELECT * FROM table_name

WHERE A=1 AND B=1;

OR & IS NULL

OR

E.g.

```
SELECT * FROM table_name
WHERE A=1 OR B=1;
```

IS NULL

E.g.

```
SELECT * FROM table_name
WHERE A IS NULL;
```

BETWEEN & IN

BETWEEN

E.g.

```
SELECT * FROM table_name
WHERE A BETWEEN 1 AND 10;
```

IN

E.g.

```
SELECT * FROM table_name
WHERE C IN ('CAKE', 'DRINK', 'NOODLES');
```

LIKE

%

```
SELECT * FROM table_name
WHERE C LIKE ‘%CAKE’;
```

_

```
SELECT * FROM table_name
WHERE C LIKE ‘_CAKE’;
```

JOIN

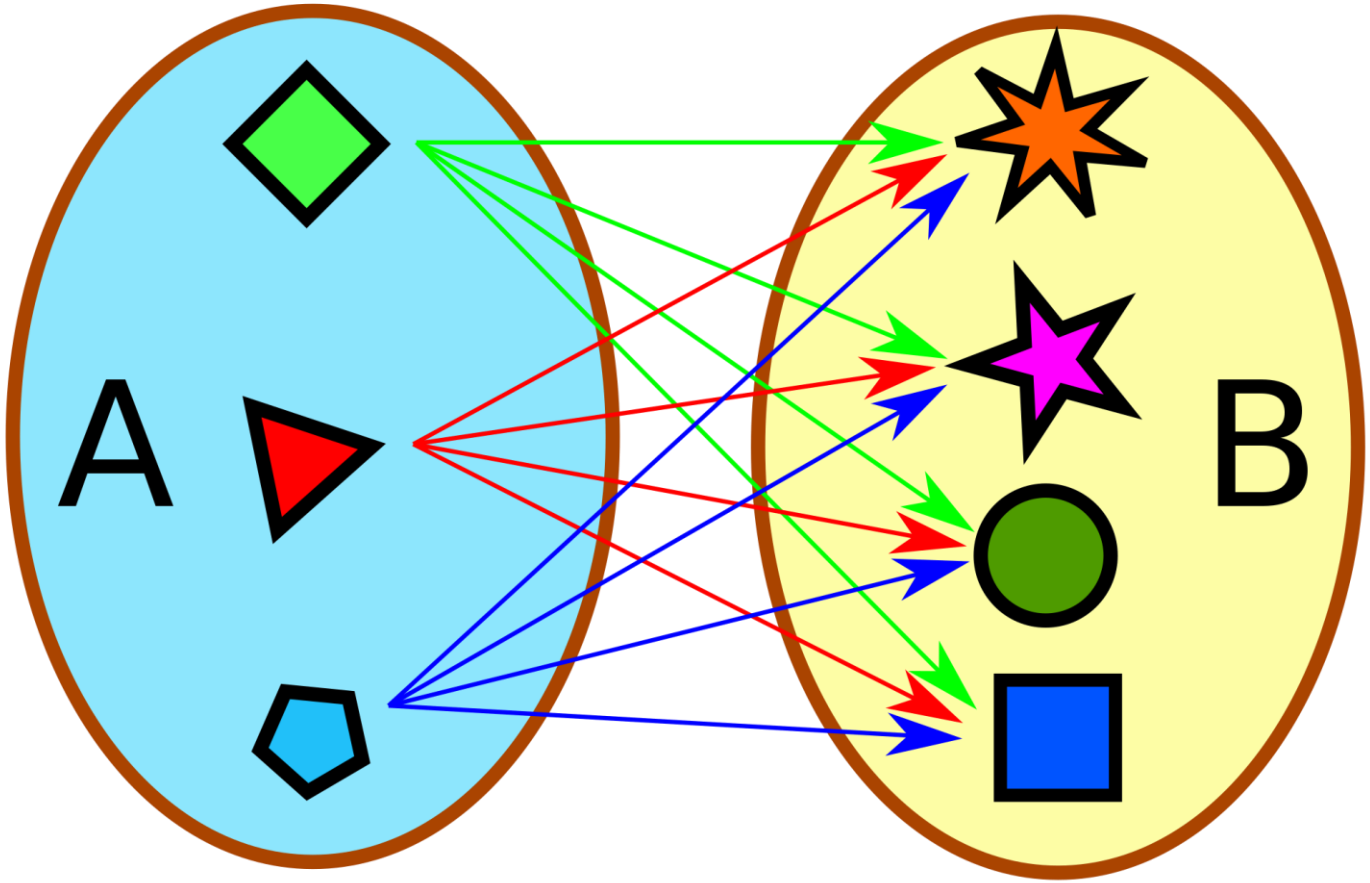
implicit join

```
SELECT * FROM tableA
WHERE tableA.id=tableB.id;
```

explicit join

```
SELECT * FROM tableA INNER JOIN tableB
ON tableA.id=tableB.id;
```

CROSS JOIN



INNER JOIN

- CROSS JOIN
- EQUIJOIN
- NON-EQUIJOIN
- NATURAL JOIN

EQUIJOIN

Equal

E.g.

```
SELECT table1.A, table2.B
FROM table1
    INNER JOIN
    table2
ON table1.id=table2.id;
```

NON-EQUIJOIN

Not equal

E.g.

```
SELECT table1.A, table2.B
FROM table1
    INNER JOIN
    table2
ON table1.id<>table2.id;
```

NATURAL JOIN

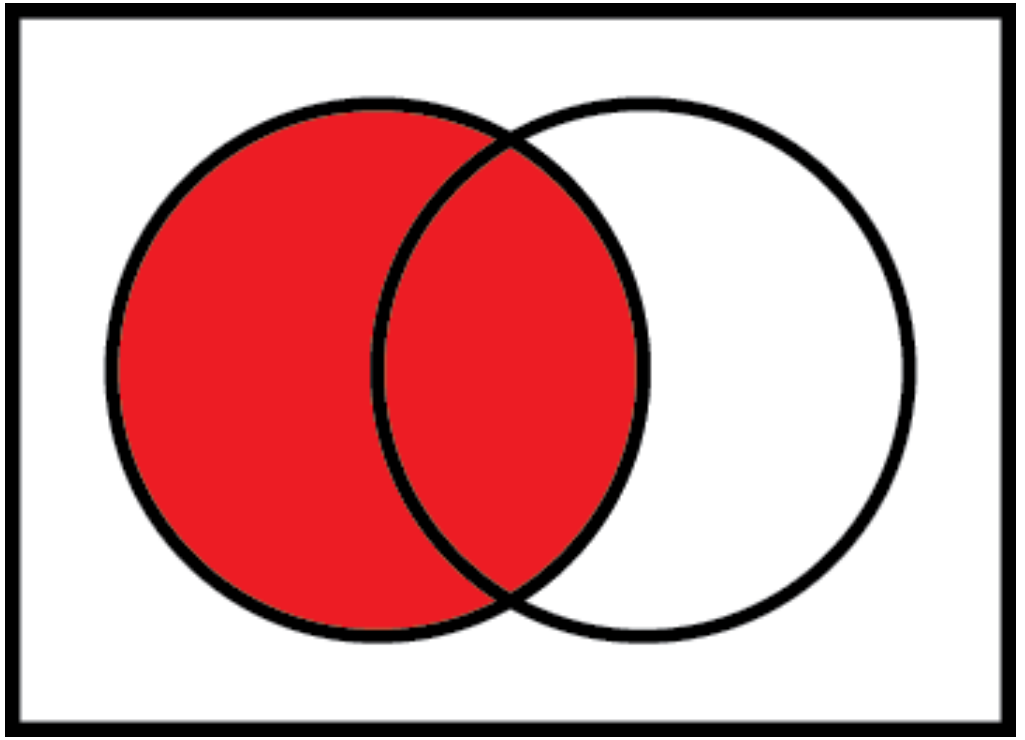
- Use the same name

```
SELECT table1.A, table2.B
FROM table1
    NATURAL JOIN
    table2;
```

OUTER JOIN

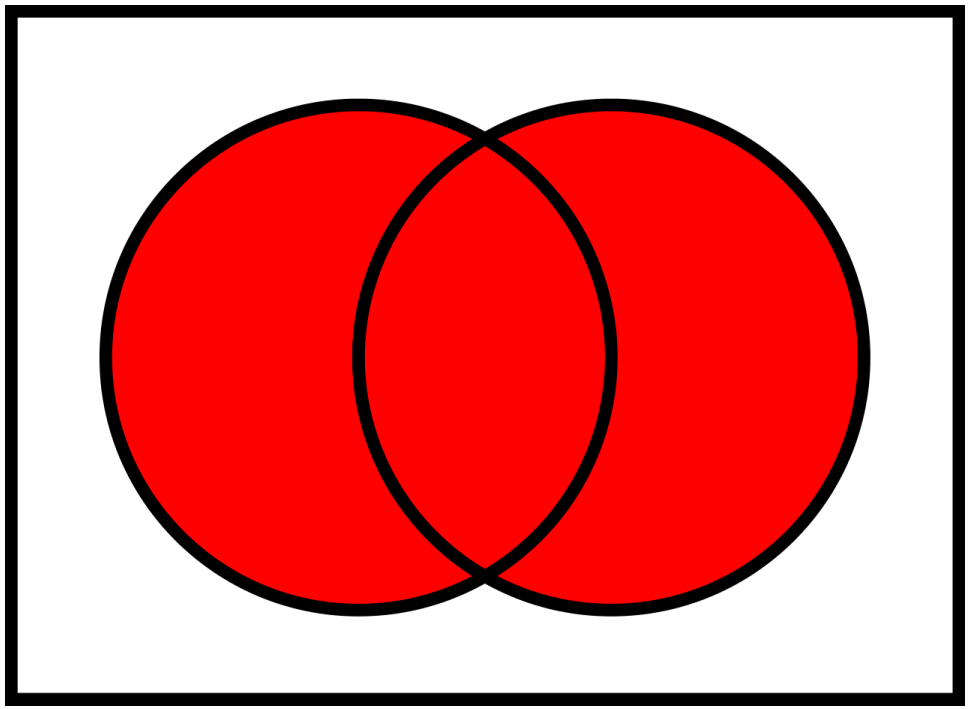
- LEFT OUTER JOIN

```
SELECT t1.A, t2.B

FROM table1 t1

LEFT OUTER JOIN table2 t2

ON t1.id=t2.id
```



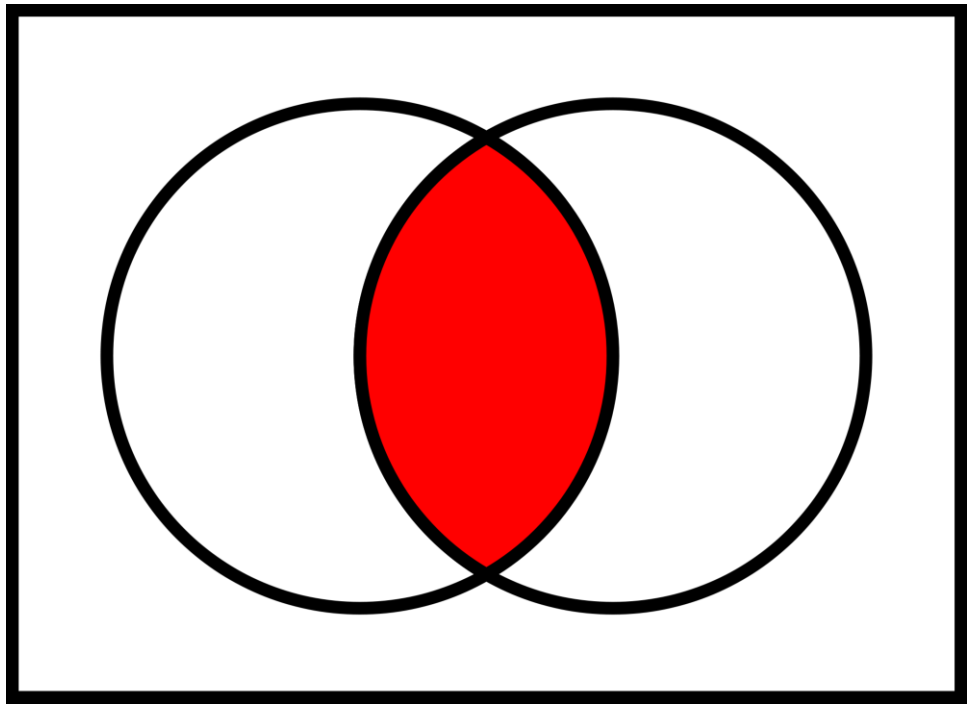
UNION

```
SELECT column_name
FROM table1
UNION
SELECT column_name
FROM table2
UNION
SELECT column_name
FROM table3;
```



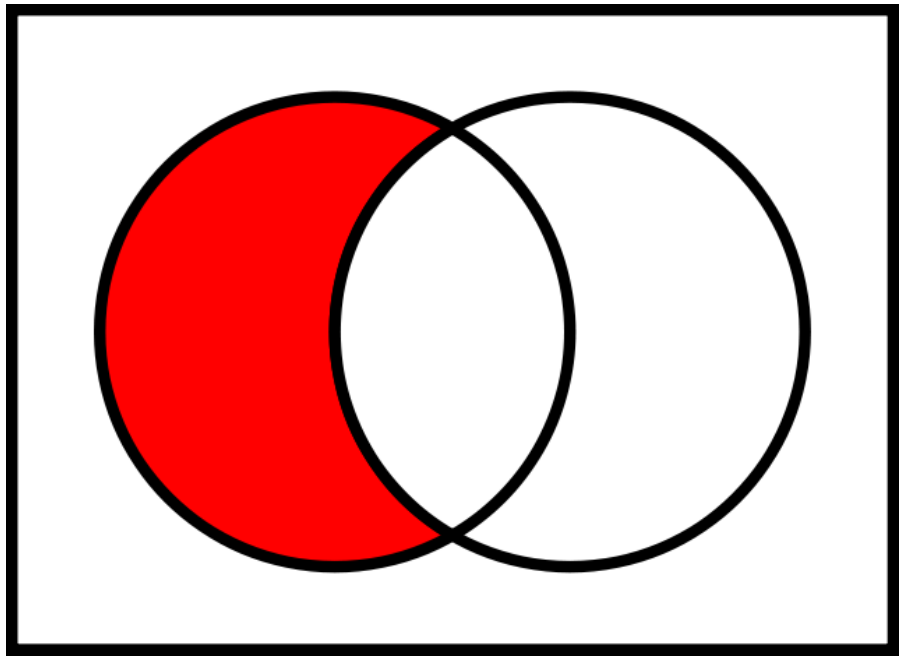
INTERSECT

```
SELECT column_name
FROM table1
INTERSECT
SELECT column_name
FROM table2;
```



EXCEPT

```
SELECT column_name FROM
table1
EXCEPT
SELECT column_name FROM
table2;
```



Aggregate Funtions

SUM, MIN, MAX, AVG, COUNT

SUM

Eg.

SELECT SUM(column_name)

FROM table_name;

COUNT

Eg.

SELECT COUNT(column_name)

FROM table;

GROUP BY

E.g.

```
SELECT column_name1, SUM(column_name2)

FROM table

GROUP BY column_name1;
```

ORDER BY

- Ascending Order

```
SELECT  column_name1 FROM table

ORDER BY column_name1;
```

- Descending Order

```
SELECT  column_name1, column_name2 FROM table

ORDER BY column_name1 ASC, column_name2 DESC;
```

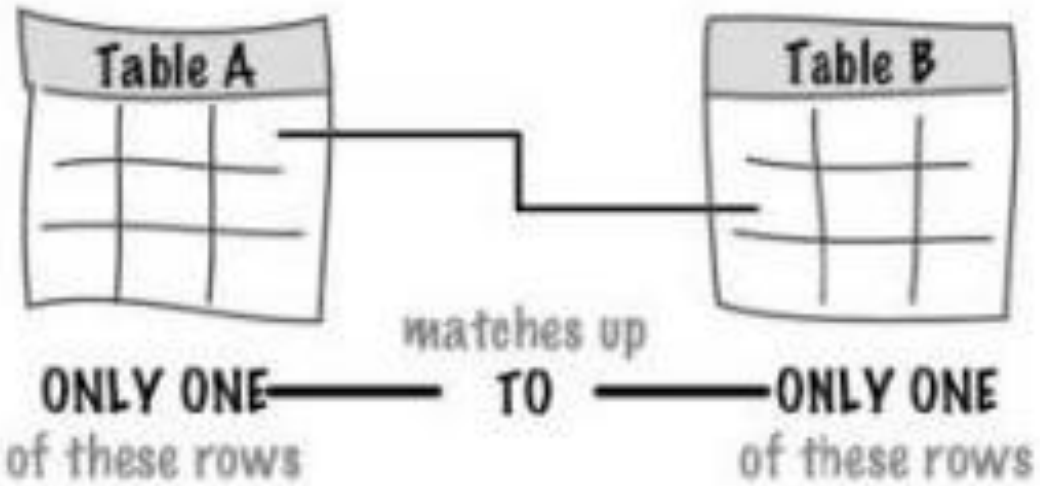
Normalization

- **What is Normalization?**
  - 讓資料表遵循某些標準規則
- **Why should we do Normalization?**
  - 重覆儲存資料會耗費硬碟空間以及增加撰寫語法複雜度的缺點
  - 減小資料庫大小，查詢較快。優化讀取時間
  - 讓新的設計師能快速瞭解資料表

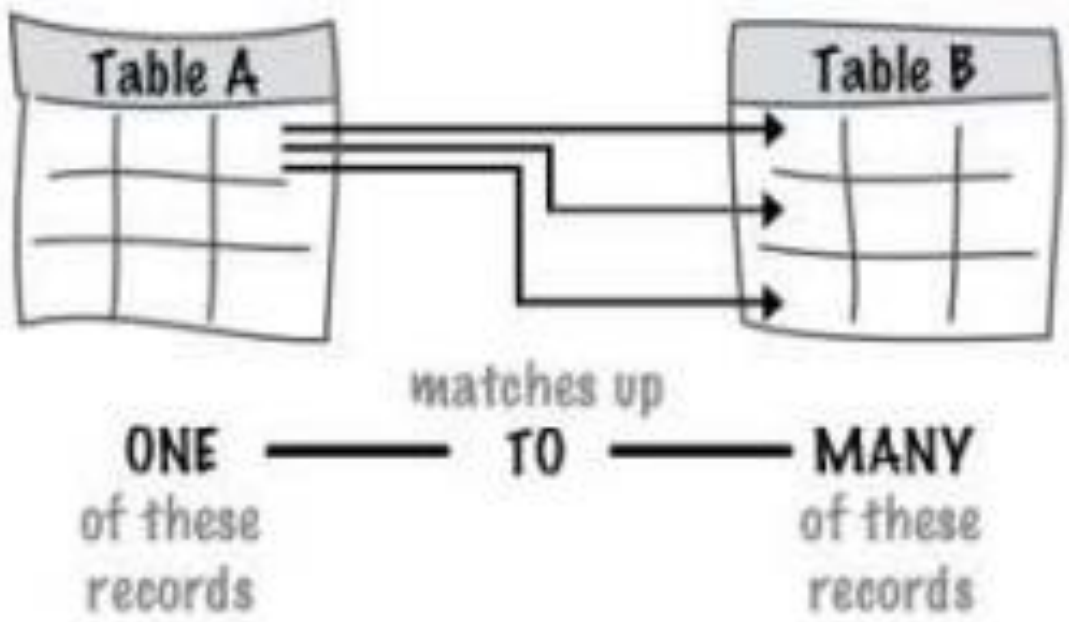
Key

- PRIMARY KEY – a column in your table that makes each record unique:
  - Unique
  - Cannot be NULL
  - Cannot be alter
- **FOREIGN KEY** – a column in a table that references the PRIMARY KEY of another table:
  - The values of foreign key must exist in the table that the key came from.
  - Can be NULL
  - Can be duplicate

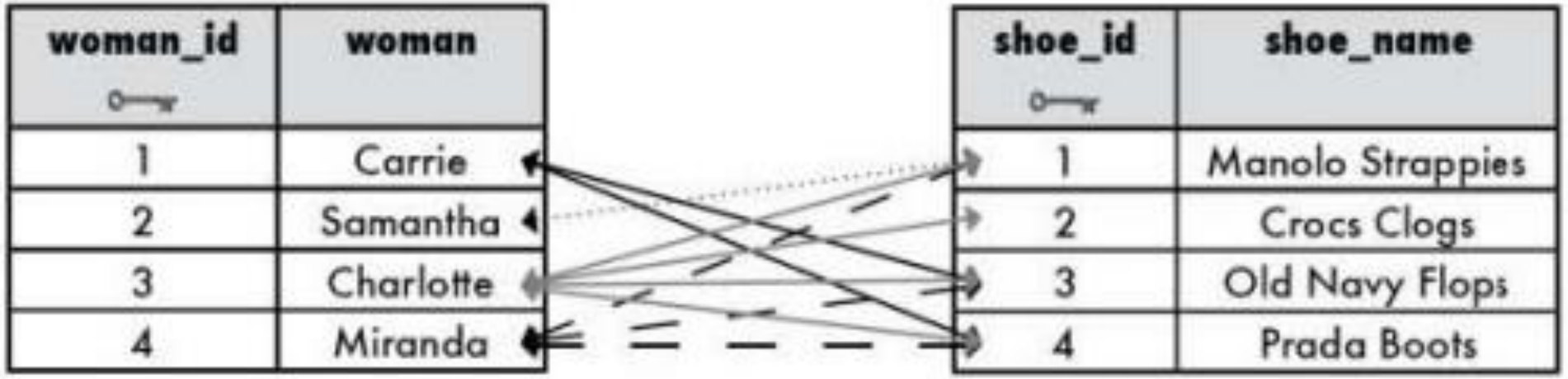
Data mode – One to One



Data mode – One to Many



Data mode – Many to Many



Third normal form (3NF)

01 單元性
Atomic

02 部份功能相依性
Partial functional dependency

03 遷移功能相依性
Transitive dependency

1NF

Meet Atomic and without duplicate data group

2NF

Meet 1NF and without Partial functional dependency

3NF

Meet 2NF and without Transitive dependency

Small Database Design

1. 定義描述的目標:
   - 描述什麼?
   - 範圍
2. 定義核心對象:
   - 把核心對象變成一張表
   - 在乎什麼資訊? 列出需要的資料
   - 想要怎麼使用這些資料表? 想要怎麼搜尋?

Small Database Design

3. 分析資料表之間的關係:
   - 資料欄之間如何產生關聯

   Eg. One to Many

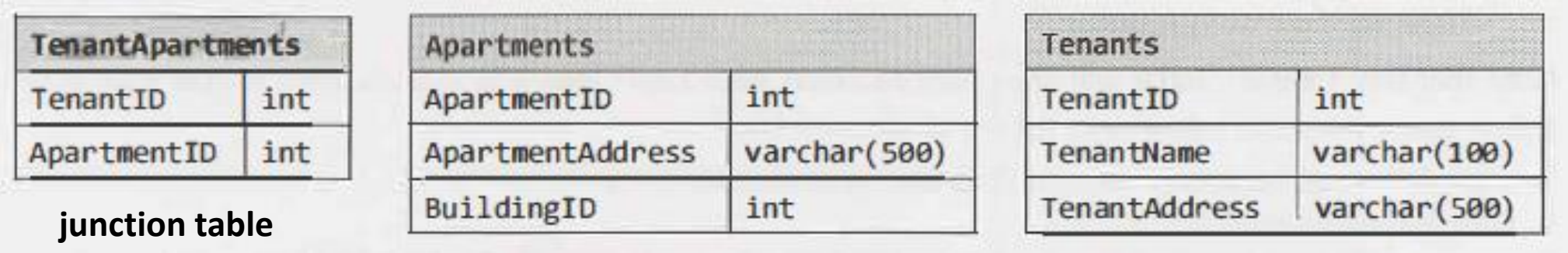
Apartments	
ApartmentID	int
ApartmentAddress	varchar(100)
BuildingID	int

Buildings	
BuildingID	int
BuildingName	varchar(100)
BuildingAddress	varchar(500)

Small Database Design

3. 分析資料表之間的關係:

Eg. Many to Many



TenantApartments	
TenantID	int
ApartmentID	int

junction table

Apartments	
ApartmentID	int
ApartmentAddress	varchar(500)
BuildingID	int

Tenants	
TenantID	int
TenantName	varchar(100)
TenantAddress	varchar(500)

4. 更好地組織資料:

- 拆解成小塊資訊
- 進行正規化

Large Database Design

- **When we use large or extendable database**:
  - Some join become slow
- **What is Denormalization?**

E.g.

  - Save duplicate data
  - Save aggregate result

Denormalization

- **Advantage**:
  - Return result more quickly when we seldom write data and read data largely
  - Reduce duplicated computing
  - Easy to cut or extend data
- **Drawback:**
  - Use more disk storage space
  - Cannot update data easily
  - Maybe need more GROUP BY or DISTINCT

Questions

Questions 1 through 3 refer to the database schema at the end of the chapter. Each apartment can have multiple tenants, and each tenant can have multiple apartments. Each apartment belongs to one building, and each building belongs to one complex.

14.1 Multiple Apartments: Write a SQL query to get a list of tenants who are renting more than one apartment.

Hints: #408

14.2 Open Requests: Write a SQL query to get a list of all buildings and the number of open requests (`Requests` in which `status` equals 'Open').

Hints: #411

pg 442

14.3 Close All Requests: Building #11 is undergoing a major renovation. Implement a query to close all requests from apartments in this building.

pg 442

Apartments	
AptID	int
UnitNumber	varchar(10)
BuildingID	int

Buildings	
BuildingID	int
ComplexID	int
BuildingName	varchar(100)
Address	varchar(500)

Requests	
RequestID	int
Status	varchar(100)
AptID	int
Description	varchar(500)

Complexes	
ComplexID	int
ComplexName	varchar(100)

AptTenants	
TenantID	int
AptID	int

Tenants	
TenantID	int
TenantName	varchar(100)

THANK YOU!

Do you have any questions?