Chapter 14 - Databases

// **Denormalized vs Normalized**

- normalized databases are designed to minimize redundancy
- denormalized databases are designed to optimize read time

// **Small Database Design**

- handle ambiguity → need to understand exactly what you need to design
- define the core objects
- analyze relationships → one-to-many, many-to-many
- investigate actions

// **Large Database Design**

- When designing a large, scalable database, joins are generally very slow
- you must denormalize your data → duplicate the data in multiple tables

// **Multiple Apratments** → get a list of tenants who are renting more than one apartment

```
SELECT TenantName FROM Tenants
INNER JOIN (
        SELECT TenantID FROM AptTenants GROUP BY TenantID HAVING count(*) > 1
) C ON Tenants.TenantID = C.TenantID
```

// **Open Requests** → get a list of all buildings and the number of open requests

```
SELECT BuildingName, ISNULL(Count, 0) as 'Count'
FROM Buildings
LEFT JOIN
        (SELECT Apartments.BuildingID, count(*) as 'Count'
         FROM Requests INNER JOIN Apartments
         ON Requests.AptID = Apartments.AptID
         WHERE Requests.Status = 'Open'
         GROUP BY Apartments.BuildingID) ReqCounts
ON ReqCounts.BuildingID = Buildings.BuildingID
```

// **Close all requests**

```
UPDATE Request
SET Status = 'Closed'
WHERE AptID IN (SELECT AptID FROM Apartments WHERE BuildingID = 11)
```

// **Joins** → used to combine the results of two tables

- to perform a JOIN each of the tables must have at least one field that will be used to find matching records from the other table
- INNER JOIN → contains only the data where criteria match
- OUTER JOIN → will contain INNER JOIN results and some records that have no matching record in the other table
    - LEFT OUTER JOIN → all records from left table, if not matching records were found in the right table, then its fields will contain the NULL values
    - RIGHT OUTER JOIN → the opposite of left JOIN
    - FULL OUTER JOIN → combines the results of a LEFT and RIGHT join

// **Denormalization**

- database optimization technique → add redundant data to one or more tables
- helps avoid costly joins
- updates and inserts are more expensive
- reads are faster
- data can be inconsistent
- data redundancy needs more storage

// **Design Grade Database**

- trade-off between flexibility and complexity