COMP163
Database Management Systems
September 18, 2008

Lecture 8 – Sections 8.4-8.5
SQL Queries
**SQL**

- **SQL** *(Structured Query Language)* is the standard language for commercial DBMSs
- **SEQUEL** *(Structured English QUEry Language)* was originally defined by IBM for **SYSTEM R**
  - mid 1970s
  - unofficial pronunciation *(see-kwuhl)* still sometimes used

- Standardization of SQL began in the 80s
- Current standard is SQL-99
  - Subsequent revisions are not fully accepted by all vendors

- SQL is more than a query language: it includes a DDL, DML and admin commands
SQL commands

- Administration:
  - CREATE DATABASE
  - CREATE SCHEMA
  - SET ROLE
  - GRANT PRIVILEGES

- Data Definition:
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
  - CREATE VIEW

- Data Modification:
  - INSERT
  - DELETE
  - UPDATE

- Queries:
  - SELECT

48 commands listed in SQL in a Nutshell
SQL Queries

- Queries in SQL are variations of the SELECT command

- Basic SQL queries correspond to the following relational algebra operations:
  - select $\sigma$
  - project $\pi$
  - cross product $X$
  - joins must be expressed as $\sigma$ and $X$
Basic SELECT Command

SELECT <attribute list>
FROM <table list>
WHERE <condition>;

project π

cross product X

select σ
Single Table Queries ($\sigma$ and $\pi$)

$$\pi_{\text{Ssn}} (\sigma_{\text{Salary} > 60000} (\text{EMPLOYEE}))$$

SELECT Ssn
FROM EMPLOYEE
WHERE Salary > 60000;

$$\pi_{\text{City}, \text{State}} (\sigma_{\text{Airport\_code} = 'SFO'} (\text{AIRPORT}))$$

SELECT City, State
FROM AIRPORT
WHERE Airport\_code = 'SFO';
Join as Select & Cross

- In the basic SELECT/FROM/WHERE form, joins must be expressed as using $\sigma$ and $\times$

\[
\pi_{\text{Lname}, \text{Dname}} \left( \text{EMPLOYEE} \bowtie_{\text{Ssn}=\text{Mgr_ssn}} \text{DEPARTMENT} \right)
\]

\[
\pi_{\text{Lname}, \text{Dname}} \left( \sigma_{\text{Ssn}=\text{Mgr_ssn}} \left( \text{EMPLOYEE} \times \text{DEPARTMENT} \right) \right)
\]

**SELECT** Lname, Dname
**FROM** EMPLOYEE, DEPARTMENT
**WHERE** Ssn = Mgr_ssn;
Basic SQL Queries

• Retrieve the name and address of all employees who work for the 'Research' department.

```sql
SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO
```

selection condition  join condition
For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

```
SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN
AND PLOCATION='Stafford'
```
We can give names to the tuples coming from each of the input relations

```
SELECT E.Lname, D.Dname
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.Ssn = D.Mgr_ssn;
```

This can disambiguate common attribute names and improve readability.
Renaming Attributes

- Attributes can also be renamed in the FROM clause
  - similar to alternate rename syntax in the algebra

```
SELECT    Fn, Ln
FROM      EMPLOYEE E(Fn, Mi, Ln, Bd, Ad, Sx, Sl, Sssn, Dn)
WHERE     Dn = 4;
```
For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```sql
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E, EMPLOYEE S
WHERE E.SUPERSSN = S.SSN
```

- Aliases are necessary for this query

- Think of E and S as two different copies of EMPLOYEE
  - E represents employees in role of supervisees and
  - S represents employees in role of supervisors
Aliases: alternate syntax

- Can also use the AS keyword to specify aliases

```sql
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME 
FROM EMPLOYEE AS E, EMPLOYEE AS S 
WHERE E.SUPERSSN=S.SSN
```

- Can also simply use the relation names (when non-ambiguous)

```sql
SELECT EMPLOYEE.Lname, DEPARTMENT.Dname 
FROM EMPLOYEE, DEPARTMENT 
WHERE EMPLOYEE.Ssn = DEPARTMENT.Mgr_ssn;
```
No $\sigma \rightarrow$ No WHERE

- If there are no selection (or join) conditions, the WHERE clause can be omitted

```
SELECT Ssn
FROM EMPLOYEE
\pi_{Ssn} EMPLOYEE
```

- Two or more relations in FROM clause with no join is a CROSS PRODUCT

```
SELECT Lname, Dname
FROM EMPLOYEE, DEPARTMENT
\pi_{Lname, Dname} (EMPLOYEE X DEPARTMENT)
```
**No π → ***

- To retrieve all the attribute values of the selected tuples, use *, which stands for *all the attributes*

```
SELECT * FROM EMPLOYEE
WHERE DNO=5
```

```
SELECT * FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNO=DNUMBER
```
Tables as Sets → DISTINCT

- SQL does not treat a relation as a set; duplicate tuples can appear.
- To eliminate duplicate tuples in a query result, the keyword `DISTINCT` is used.

```
SELECT SALARY FROM EMPLOYEE  # may contain duplicates

SELECT DISTINCT SALARY FROM EMPLOYEE  # duplicates eliminated
```
Set Operations

- union operation (UNION)
- intersection (INTERSECT)
- set difference (MINUS, sometimes called EXCEPT)
  - some implementations of SQL do not support all set operations

- Set operation results are sets of tuples
  duplicate tuples are eliminated from the result

- The set operations apply only to union compatible relations:
  the two relations must have the same attributes and
  the attributes must appear in the same order
Set Operations: Example

- List project numbers for all projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

\[(\text{SELECT} \ PNAME \ \text{FROM} \ PROJECT, \ DEPARTMENT, \ EMPLOYEE \ \text{WHERE} \ DNUM=\text{DNUMBER} \ \text{AND} \ \text{MGRSSN}=\text{SSN} \ \text{AND} \ \text{LNAME}=\text{'Smith'} )\]

UNION

\[(\text{SELECT} \ PNAME \ \text{FROM} \ PROJECT, \ WORKS\_\text{ON}, \ EMPLOYEE \ \text{WHERE} \ PNUMBER=\text{PNO} \ \text{AND} \ \text{ESSN}=\text{SSN} \ \text{AND} \ \text{NAME}=\text{'Smith'} )\]
Multiset Operations

- UNION ALL, INTERSECT ALL, EXCEPT ALL
- Multiset operation results are multisets of tuples
  *duplicate tuples are not eliminated*

(_SELECT PNAME
  FROM PROJECT, DEPARTMENT, EMPLOYEE
  WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='Smith')
UNION ALL
(_SELECT PNAME
  FROM PROJECT, WORKS_ON, EMPLOYEE
  WHERE PNUMBER=PNO AND ESSN=SSN AND NAME='Smith')
WHERE Clause

- WHERE clause is a general boolean expression
- Boolean operators: AND, OR, NOT
- Comparison operators: =, <, <=, >, >=, <>
- String comparison operators: LIKE
- Parentheses can be used to set precedence
- String literals can be enclosed in "..." or '...'
String Comparison

- The **LIKE** comparison operator is used to compare partial strings.

- Two wildcard characters are used:
  
  '\%' replaces an arbitrary number of characters

  '_.' replaces a single arbitrary character
String Comparison Example

- Retrieve all employees whose address is in Houston, Texas.

- The value of the ADDRESS attribute must contain the substring “Houston, TX”.

```sql
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE ADDRESS LIKE '%Houston, TX%'
```

zero or more characters, before and after substring
String Comparison Example

- Retrieve all employees who were born during the 1960s.
  - ‘6' must be the 3rd character of the 10 character date string

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE BDATE LIKE '__6____________'
```

- Following would also work:

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE BDATE LIKE '__6 %'
```

assumes date format is YYYY-MM-DD
Arithmetic Operation

- The standard arithmetic operators '+', '-', '*', and '/' can be applied to numeric values in an SQL query result.

- Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```sql
SELECT FNAME, LNAME, 1.1*SALARY
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN=ESSN AND PNO=PNUMBER
AND PNAME='ProductX'
```
Aggregate Functions

- Aggregate functions are applied to result attributes \texttt{COUNT}, \texttt{SUM}, \texttt{MAX}, \texttt{MIN}, and \texttt{AVG}

- Find the maximum salary, the minimum salary, and the average salary among all employees.

  
  \begin{verbatim}
  SELECT MAX(Salary), MIN(Salary), AVG(Salary)
  FROM EMPLOYEE
  \end{verbatim}

- Find the total salary paid to employees who work for the 'Research' department.

  \begin{verbatim}
  SELECT SUM(Salary)
  FROM EMPLOYEE, DEPARTMENT
  WHERE Dno=Dnumber AND Dname='Research'
  \end{verbatim}
Aggregate Functions

- Retrieve the total number of employees in the company and the number of employees in the Research' department.

```
SELECT COUNT (*)
FROM EMPLOYEE

SELECT COUNT (*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```
Join as $X$ and $\sigma$

mysql> SELECT * FROM r;
+-----+-----+
| x   | y   |
+-----+-----+
| 3   | 4   |
| 5   | 6   |
| 7   | 8   |
| 9   | 6   |
+-----+-----+

mysql> SELECT * FROM s;
+-----+-----+
| a   | b   |
+-----+-----+
| 2   | 3   |
| 4   | 7   |
+-----+-----+

mysql> SELECT * FROM r,s WHERE y=a;
+-----+-----+-----+-----+
| x   | y   | a   | b   |
+-----+-----+-----+-----+
| 3   | 4   | 4   | 7   |
+-----+-----+-----+-----+

R $\bowtie$ y=a S

$\sigma$ $y=a$ (R $\times$ S)
Explicit Join

- Joins can be explicitly stated in the FROM clause.

```
SELECT * FROM (r JOIN s ON y=a);
+--------------------------+
| x | y | a | b  |
+--------------------------+
| 3 | 4 | 4 | 7  |
+--------------------------+
```

R  ⋈  y=a  S
### Left/Right Outer Join

**SELECT * FROM (r LEFT JOIN s ON y=a);**

+---+------+------+------+ | x | y    | a    | b    | +---+------+------+------+ | 3 | 4    | 4    | 7    | | 5 | 6    | NULL | NULL | | 7 | 8    | NULL | NULL | | 9 | 6    | NULL | NULL | | 9 | 6    | NULL | NULL | +---+------+------+------+**

**SELECT * FROM (r RIGHT JOIN s ON y=a);**

+-+---+--------+---------+ | x | y | a | b | +-------+---+--------+---------+ | NULL | NULL | 2 | 3 | | 3 | 4 | 4 | 7 | +-------+---+--------+---------+
Full Outer Join

SELECT * FROM r FULL OUTER JOIN s ON y=a;

mysql doesn’t support full outer join, so we’ll substitute an equivalent query:

(SELECT * FROM r LEFT JOIN s ON y=a) UNION
(SELECT * FROM r RIGHT JOIN s ON y=a);

+------+------+------+------+| x    | y    | a    | b    | +------+------+------+------+| 3    | 4    | 4    | 7    | | ... | | 7    | 8    | NULL | NULL | | 9    | 6    | NULL | NULL | | NULL | NULL | 2    | 3    | +------+------+------+------+
Ordering Results

- An ORDER BY clause can be added to order the result tuples

```
SELECT * FROM t;
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ten</td>
</tr>
<tr>
<td>11</td>
<td>eleven</td>
</tr>
<tr>
<td>20</td>
<td>twenty</td>
</tr>
<tr>
<td>4</td>
<td>four</td>
</tr>
</tbody>
</table>

SELECT * FROM t ORDER BY j;
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>eleven</td>
</tr>
<tr>
<td>4</td>
<td>four</td>
</tr>
<tr>
<td>10</td>
<td>ten</td>
</tr>
<tr>
<td>20</td>
<td>twenty</td>
</tr>
</tbody>
</table>

SELECT * FROM t ORDER BY i;
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>four</td>
</tr>
<tr>
<td>10</td>
<td>ten</td>
</tr>
<tr>
<td>11</td>
<td>eleven</td>
</tr>
</tbody>
</table>
| 20 | twenty |```
ORDER BY Examples

- order by Lname first,
  then by Fname if Lname is the same:

  SELECT Lname, Fname
  FROM Employee
  WHERE salary > 60000
  ORDER BY Lname, Fname

- order by Lname in ascending order, then by salary in descending order

  SELECT Lname, salary
  FROM Employee
  WHERE salary > 60000
  ORDER BY Lname ASC, salary DESC
Grouping

- Forms groups (subsets) of result tuples before applying aggregate functions
- Example: count the number of employees in each department
  (group employees by DNO, then count tuples in each group)

```
SELECT Dno, COUNT(*)
FROM Employee
GROUP BY Dno
```

<table>
<thead>
<tr>
<th>Dno</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>120</td>
</tr>
<tr>
<td>22</td>
<td>238</td>
</tr>
<tr>
<td>7</td>
<td>82</td>
</tr>
<tr>
<td>20</td>
<td>169</td>
</tr>
</tbody>
</table>
```
GROUP BY Example

- For each project, get the project name, project number and the number of employees working on that project

```
SELECT Pnumber, Pname, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE Pnumber = Pno
GROUP BY Pnumber, Pname
```

Attributes in SELECT clause must be aggregates or must appear in the GROUP BY clause
Filtering Groups: HAVING

- We can throw away some groups by adding a condition in a HAVING clause.

- example:
  for each project that has more than two employees, get the project name, project number and the number of employees working on that project.

```sql
SELECT Pnumber, Pname, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE Pnumber = Pno
GROUP BY Pnumber, Pname
HAVING COUNT(*) > 2
```
GROUP BY Examples

```
SELECT COUNT(*) FROM e
GROUP BY dept;

<table>
<thead>
<tr>
<th>count(*)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```

```
SELECT * FROM e;

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E01</td>
<td>65000</td>
<td>ADMIN</td>
</tr>
<tr>
<td>E12</td>
<td>58400</td>
<td>ENGR</td>
</tr>
<tr>
<td>E08</td>
<td>76900</td>
<td>ENGR</td>
</tr>
<tr>
<td>E23</td>
<td>63800</td>
<td>ADMIN</td>
</tr>
<tr>
<td>E07</td>
<td>56900</td>
<td>ADMIN</td>
</tr>
<tr>
<td>E27</td>
<td>76400</td>
<td>ENGR</td>
</tr>
<tr>
<td>E14</td>
<td>48000</td>
<td>TEST</td>
</tr>
</tbody>
</table>
```

```
SELECT dept, COUNT(*) FROM e GROUP BY dept;

<table>
<thead>
<tr>
<th>dept</th>
<th>count(*)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMIN</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>ENGR</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```

```
SELECT dept, COUNT(*) FROM e GROUP BY dept;

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMIN</td>
<td>3</td>
</tr>
<tr>
<td>ENGR</td>
<td>3</td>
</tr>
<tr>
<td>TEST</td>
<td>1</td>
</tr>
</tbody>
</table>
```
GROUP BY Examples

```
SELECT dept, COUNT(*)
FROM e
GROUP BY dept
HAVING COUNT(*) > 1;
```

```
+-------+----------+
| ADMIN |        3 |
| ENGR  |        3 |
+-------+----------+

```

```
SELECT dept, AVG(salary)
FROM e
GROUP BY dept
HAVING COUNT(*) > 1;
```

```
<table>
<thead>
<tr>
<th>dept</th>
<th>AVG(salary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMIN</td>
<td>61900</td>
</tr>
<tr>
<td>ENGR</td>
<td>70566.66667</td>
</tr>
</tbody>
</table>
+-------+-------------+
```
Nested Queries

- Nested queries can be used as set values in the WHERE clause.

- Set comparison operators:
  - **IN** – set membership (“is in”, ∈)
  - **EXISTS** – set not empty (∃)
  - **ALL** – applies to all set members (∀)
  - **ANY** – applies to any set member
  - **CONTAINS** – proper superset
Nested Queries

- find all employees who work on a project with John Smith

```sql
SELECT Lname, Fname
FROM EMPLOYEE E1, WORKS_ON W1
WHERE E1.SSN = W1.ESSN
AND W1.Pno IN (SELECT Pno
                 FROM EMPLOYEE E2, WORKS_ON W2
                 WHERE E2.SSN = W2.ESSN
                 AND E2.Fname = "John"
                 AND E2.Lname = "Smith")
```
Nested Queries

- find the highest paid employee in department 5

```
SELECT Lname, Fname
FROM EMPLOYEE E1
WHERE E1.Dno=5
    AND E1.Salary > ALL (SELECT E2.Salary
                           FROM EMPLOYEE E2
                           WHERE E2.Dno=5)
```
Nested Queries

• List names of managers who have dependents

SELECT Lname, Fname
FROM EMPLOYEE E1
WHERE EXISTS (SELECT *
FROM DEPENDENT D1
WHERE E1.Ssn = D1.Essn)
AND
EXISTS (SELECT *
FROM DEPARTMENT D2
WHERE E1.Ssn = D2.Mgr_ssn)

This is an example of a correlated nested query, since the nested queries refer to the relations in the outer query.
Nested Queries

- List names of employees who work on all projects controlled by department 5

```sql
SELECT Lname, Fname
FROM EMPLOYEE E
WHERE (SELECT W.Pno
      FROM WORKS_ON W
      WHERE E.Ssn = W.Essn)
  CONTAINS
  (SELECT P.Pnumber
   FROM PROJECT P
   WHERE P.Dnum=5)
```
Nested Queries

- List names of all projects controlled by department 5 or department 7

```
SELECT P.Pname
FROM PROJECT P
WHERE P.Dnum IN (5,7)
```
SELECT: Syntax Summary

SELECT <attribute and function list>
FROM <table list>
WHERE <condition>
GROUP BY <grouping attributes>
HAVING <group condition>
ORDER BY <attribute list>

required

optional
SELECT: conceptual execution

1. FROM: cross product of tables
2. WHERE: select tuples
3. GROUP BY: group tuples
4. HAVING: filter groups
5. SELECT: project attributes and apply aggregates
6. ORDER BY: sort the tuples

This is not an efficient way to execute the query, simply a way to define the meaning of the query conceptually.
EXERCISE 1: Queries

1. First and last name of employees who have no supervisor.
2. First and last name of employees supervised by Franklin Wong.
3. Last name of employees who have dependents.
4. Last name of employees who have daughters.
5. Last name of employees in department 5 who work more than 10 hours/week on ProductX.
6. Last name of supervisors of employees in department 5 who work more than 10 hours/week on ProductX.
7. First and last names of all department managers.
8. Salaries of all employees who have worked on the Reorganization project.
9. SSN of all employees who have worked on a project that is controlled by a department different than the department that they are assigned to.
10. Last name of all employees who are not married.
EXERCISE 2: Queries

1. List all airplane types that can land at any airport in San Francisco.
2. List the ids and number of seats for all airplanes that can land at any airport in Chicago.
3. List the name and phone number of all customers with a seat reserved on a flight that leaves Chicago O’Hara airport (ORD) on October 31, 2008.
4. List all airlines that have seats available for flights leaving Los Angeles (LAX) on September 25, 2008.
5. List all airlines that operate at San Jose International Airport (SJC).
EXERCISE 3: Queries

1. Count the number of overdue books.

2. How many books by author Harry Crews are in the database?

3. Determine the number of library cards assigned to each borrower phone number.

4. Find names of all borrowers who do not have any book loans.

5. Do any library branches have every book?
EXERCISE 1: Schema

| EMPLOYEE
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fname</td>
</tr>
</tbody>
</table>

| DEPARTMENT
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dname</td>
</tr>
</tbody>
</table>

| DEPT_LOCATIONS
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dnumber</td>
</tr>
</tbody>
</table>

| PROJECT
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pname</td>
</tr>
</tbody>
</table>

| WORKS_ON
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Essn</td>
</tr>
</tbody>
</table>

| DEPENDENT
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Essn</td>
</tr>
</tbody>
</table>
EXERCISE 1: Instance
## Exercise 2: Schema

### AIRPORT

<table>
<thead>
<tr>
<th>Airport_code</th>
<th>Name</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
</table>

### FLIGHT

<table>
<thead>
<tr>
<th>Flight_number</th>
<th>Airline</th>
<th>Weekdays</th>
</tr>
</thead>
</table>

### FLIGHT_LEG

<table>
<thead>
<tr>
<th>Flight_number</th>
<th>Leg_number</th>
<th>Departure_airport_code</th>
<th>Scheduled_departure_time</th>
<th>Arrival_airport_code</th>
<th>Scheduled_arrival_time</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Departure_airport_code</th>
<th>Departure_time</th>
<th>Arrival_airport_code</th>
<th>Arrival_time</th>
</tr>
</thead>
</table>

### LEG_INSTANCE

<table>
<thead>
<tr>
<th>Flight_number</th>
<th>Leg_number</th>
<th>Date</th>
<th>Number_of_available Seats</th>
<th>Airplane_id</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Departure_airport_code</th>
<th>Departure_time</th>
<th>Arrival_airport_code</th>
<th>Arrival_time</th>
</tr>
</thead>
</table>

### FARE

<table>
<thead>
<tr>
<th>Flight_number</th>
<th>Fare_code</th>
<th>Amount</th>
<th>Restrictions</th>
</tr>
</thead>
</table>

### AIRPLANE_TYPE

<table>
<thead>
<tr>
<th>Airplane_type_name</th>
<th>Max_seats</th>
<th>Company</th>
</tr>
</thead>
</table>

### CAN_LAND

<table>
<thead>
<tr>
<th>Airplane_type_name</th>
<th>Airport_code</th>
</tr>
</thead>
</table>

### AIRPLANE

<table>
<thead>
<tr>
<th>Airplane_id</th>
<th>Total_number_ofSeats</th>
<th>Airplane_type</th>
</tr>
</thead>
</table>

### SEAT_RESERVATION

<table>
<thead>
<tr>
<th>Flight_number</th>
<th>Leg_number</th>
<th>Date</th>
<th>Seat_number</th>
<th>Customer_name</th>
<th>Customer_phone</th>
</tr>
</thead>
</table>
EXERCISE 3: Schema

- **BOOK**
  - Book_id
  - Title
  - Publisher_name

- **BOOK_AUTHORS**
  - Book_id
  - Author_name

- **PUBLISHER**
  - Name
  - Address
  - Phone

- **BOOK_COPIES**
  - Book_id
  - Branch_id
  - No_of_copies

- **BOOK_LOANS**
  - Book_id
  - Branch_id
  - Card_no
  - Date_out
  - Due_date

- **LIBRARY_BRANCH**
  - Branch_id
  - Branch_name
  - Address

- **BORROWER**
  - Card_no
  - Name
  - Address
  - Phone