Week 5

Relational Databases

The Relational Model, Relational Algebra, and SQL

Today’s Topics

- **Part I**:
  - Introduction to the **relational model** of database design
  - Introduction to **relational algebra**, which forms the basis of the **Structured Query Language** (SQL)
  - Mapping relational algebra to SQL
- **Part II**:
  - **Hands-on exercise** with SQL

Database Options

Repeated from Week 4

SQL databases (or relational databases)

- Well defined **schema**
  - the database is modeled as a set of joinable tables with pre-defined columns
- Entities are stored in **tables** (called relations), managed by an **RDBMS**
- **Tables can be joined** and using a powerful and flexible querying language: **SQL**
- Data follows a **rigid structure** and is mostly **non-redundant**
- Changing/updating the structure of data involves change in schema and **database alterations**.
- There is a DBA – a **database administrator** -- between the application developer and the database, responsible for defining and altering the schema if needed.
  - **Using SQL's DDL**

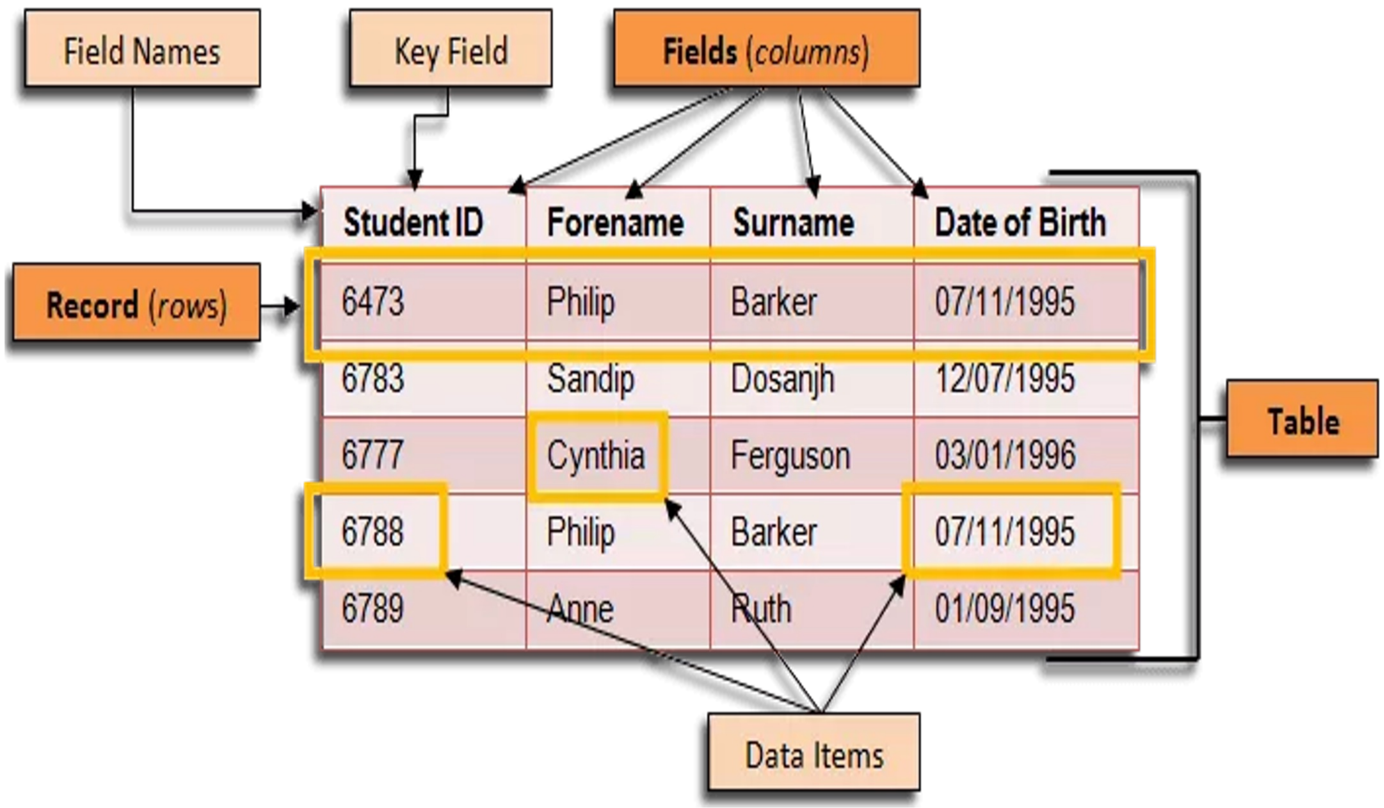
Naturally ideal for vertical scaling, i.e., increasing the database's capacity by adding more resources (CPU, memory, storage) to a single machine.

The Relational Model

A **relation** (table) contains **records** (rows) made of **fields** (columns).

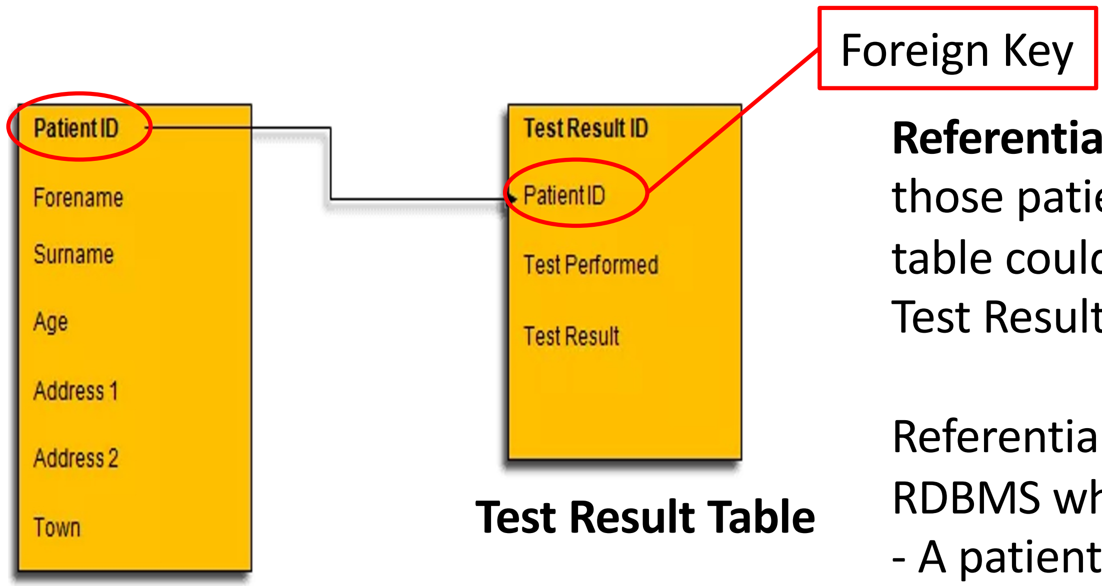
At least one of the fields is a **primary key** (key field), used to uniquely identify a record.

Important rule: each data item has a **type** (the type of that field) and must be atomic, i.e., it cannot be a list or any other kind of collection.



The Relational Model

Foreign Key: a field (or set of fields) in one relation that references the primary key of another relation, establishing a relationship and enforcing **referential integrity** between the two relations.



Patient Table

Test Result Table

Referential integrity is enforced, as only those patient IDs that exist in the Patient table could appear as foreign keys in the Test Result table

Referential Integrity is enforced by an RDBMS when, for example:
- A patient is removed from the Patient table.
- A test result is added to the Test Result table.

Relational Schema

- The name of a relation and its fields together is called the **schema** of that relation.

  e.g. the schema for a Movies relation may be written as:

  Movies (<u>title</u>, <u>year</u>, length, genre)

  - The Movies table contains rows each of which have 4 columns, namely: title, year, length and genre (usually displayed in that order).
  - title and year together are the **primary key (composite)**, i.e. the (title, year) combination uniquely identifies a row of the Movies table

- The collective schema of all relations of a database make up the **database schema**.
- **In practice,** to create a relational database, a database administrator must specify the database scheme using SQL's DDL (data definition language).

Example: A simple HR database

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

Overall description:

Employees work for departments.
Departments run projects.
Employees work on projects.
Employees have dependents.

Foreign keys:
Employee.dno
Employee.superid
Department.mgrid
Project.dno
WorksOn.empid
WorksOn.pno
Dependent.empid

Relational Operators

- **Relational operators** operate on existing relations to produce new relations corresponding to user queries

- Relational operators maybe divided into two main categories:

Set Operators	**Database Operators**
Union Intersection Difference Cartesian Product	Select Project Join Aggregate Division Rename

Projection

Extracts certain columns from the table:

$$Temp1 = \prod_{A}(r)$$

$$Temp2 = \prod_{B,C}(r)$$

r	A	B	C
	1	610	3
	1	620	3
	1	600	2
	1	650	2
	2	610	3
	2	634	4

Temp1	A
	1
	2

Set Semantics

Temp2	B	C
	610	3
	620	3
	600	2
	650	2
	634	4

Corresponding SQL:

SELECT A FROM r;

SELECT B, C FROM s;

Bag Semantics (requires DISTINCT for set semantics)

Union

- Written as (r+s) or (r U s)
- Relations must be union compatible
- Duplicate rows are eliminated

r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	A	B	C
	1	2	3
	1	1	1
	3	2	1

r + s	A	B	C
	1	1	1
	2	2	2
	3	3	3
	1	2	3
	3	2	1

Corresponding SQL:

```
SELECT * FROM r
UNION
SELECT * FROM s;
```

Intersection

- Written as (r ∩ s)
- Relations must be intersection compatible

r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	A	B	C
	1	2	3
	1	1	1
	3	2	1

r ∩ s	A	B	C
	1	1	1

Corresponding SQL:

```
SELECT * FROM r
INTERSECT
SELECT * FROM s;
```

Difference

- Written as (r-s)
- Relations have to be difference compatible
- Includes rows that are in r but not in s

r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	A	B	C
	1	2	3
	1	1	1
	3	2	1

r - s	A	B	C
	2	2	2
	3	3	3

Corresponding SQL:

```
SELECT * FROM r
EXCEPT
SELECT * FROM s;
```

Queries Using Union, Intersection, Difference and Projection

List the ids of all employees who are working on some project.

$$ans = \prod_{empid}(\text{WorksOn})$$

```
SELECT empid
FROM WorksOn;
```

List the ids of all employees who are not working on any project.

$$ans = \prod_{empid}(\text{Employee}) - \prod_{empid}(\text{WorksOn})$$

```
SELECT empid FROM Employee
EXCEPT
SELECT empid FROM WorksOn;
```

List the ids of all employees who are working on a project and have a dependent.

$$ans = \prod_{empid}(\text{WorksOn}) \cap \prod_{empid}(\text{Dependent})$$

```
SELECT empid FROM WorksOn
INTERSECT
SELECT empid FROM Dependent;
```

Schema

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

Queries Using Union, Intersection, Difference and Projection

List the ids of all supervisors who either have dependents or are working on a project or both.

$$t1 = \prod_{\text{empid}}(\text{WorksOn}) \cap \prod_{\text{superid}}(\text{Employee})$$

these are the supervisors who are working on a project

$$t2 = \prod_{\text{superid}}(\text{Employee}) \cap \prod_{\text{empid}}(\text{Dependent})$$

these are are the supervisors who have dependents

$$ans = t1 \cup t2$$

Schema

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

```
SELECT empid FROM WorksOn
INTERSECT
SELECT superid FROM Employee

UNION

SELECT superid FROM Employee
INTERSECT
SELECT empid FROM Dependent;
```

Corresponding SQL

Conditional Selection

The selection operator extracts certain rows from the table and discards the others. Retrieved tuples must satisfy a given **filtering condition**.

$$Temp1 = \sigma_{(B >= 620)\ and\ (c < 4)}\ (r)$$

r	A	B	C
	1	610	3
	1	620	3
	1	600	2
	1	650	2
	2	610	3
	2	634	4

Temp 1	A	B	C
	1	620	3
	1	650	2

```
SELECT *
FROM r
WHERE B >= 620 AND C < 4;
```

Corresponding SQL

Queries Using Union, Intersection, Difference, Projection and Conditional Selection

List the names and genders of all employees with a salary of at least 80, 000 pounds.

$$ans = \prod_{\text{fname,mint,lname,gender}} (\sigma_{salary \geq 80000}(Employee))$$

```
SELECT fname, mint, lname, gender
FROM Employee
WHERE salary >= 80000;
```

Schema

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

List the ids of all employees with a salary of at least 80, 000 pounds who are not working on any project.

$$ans = \prod_{\text{empid}} (\sigma_{salary \geq 80000}(Employee)) - \prod_{\text{empid}} (\text{WorksOn})$$

```
SELECT empid
FROM EMPLOYEE
WHERE salaray >= 80000

EXCEPT

SELECT empid
FROM WorksOn
```

Cross Product (or Cross Join)

- Written as (r x s)
- Concatenates rows from two relations, making all possible combinations of rows.

r2	A	B	C
	1	1	1
	2	2	2
	3	3	3

s2	D	E
	10	a
	20	b

r2 x s2	A	B	C	D	E
	1	1	1	10	a
	1	1	1	20	b
	2	2	2	10	a
	2	2	2	20	b
	3	3	3	10	a
	3	3	3	20	b

```
SELECT *
FROM r2, s2;
```

Corresponding SQL

Join (or inner join)

- The join operation, denoted by **(r $\bowtie_{COND}$ s)**, is used to combine *related tuples* from two relations.
- Here COND is the matching condition **r** $\bowtie_{COND}$ **s**
- The following example demonstrates the operation **r** $\bowtie_{C=D}$ **s**

r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	D	E
	1	a
	2	b
	2	c

Temp1	A	B	C	D	E
	1	1	1	1	a
	2	2	2	2	b
	2	2	2	2	c

```
SELECT *
FROM r JOIN s
ON r.C = s.D;
```

Corresponding SQL

Natural Join

Denoted by **(r * s)**

Combines tuples of two relations using an implicit condition, i.e. the tables are related by columns that have the same names and types.

r	A	B	C
	1	1	1
	2	1	0
	4	3	2

s	B	C	D
	1	1	a
	1	2	b
	3	2	c
	4	3	d

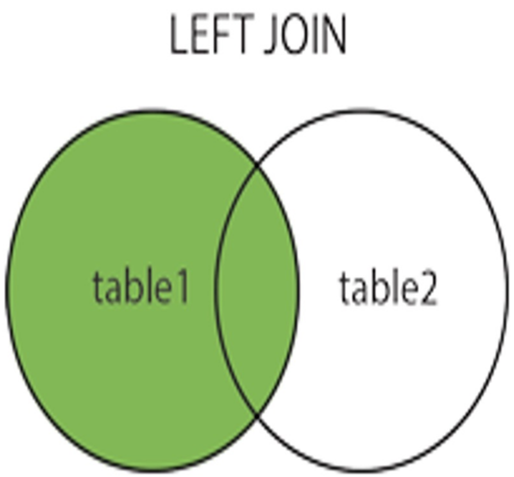
Temp	A	B	C	D
	1	1	1	a
	4	3	2	c

```
SELECT *
FROM r NATURAL JOIN s;
```

Corresponding SQL

Left Outer Join

- Denoted **(r** $⟕_{\text{COND}}$ **s)**
- Combines tuples of two relations and keeps in the result every tuple from the left table, but only those from the right table that meet the join condition.




```
SELECT *
FROM r LEFT JOIN s
ON r.A = s.D;
```

Corresponding SQL

Left Outer Join Example

$(r ⟕_{A=D} s)$

r	A	B	C
	1	1	1
	2	2	2
	3	3	3

s	D	E
	1	a
	2	b
	2	c

Temp1	A	B	C	D	E
	1	1	1	1	a
	2	2	2	2	b
	2	2	2	2	c
	3	3	3	null	null

```
SELECT *
FROM r LEFT JOIN s
ON r.A = s.D;
```

Corresponding SQL

Queries Using Set Operations, Projection, Conditional Selection and Joins

Schema

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

List the names, genders and department names of all employees.

$$ans = \prod_{\text{fname,mint,lname,gender,dname}} (\text{Employee} * \text{Department})$$

```
SELECT fname, mint, lname, gender, dname
FROM Employee NATURAL JOIN Department;
```

List the names of all the female employees in the HR department.

$$\text{ans} = \prod_{\text{fname,mint,lname}} (\sigma_{\text{gender=“F”}}(\text{Employee}) * \sigma_{\text{dname=“HR”}}(\text{Department}))$$

alternatively,

$$\text{ans} = \prod_{\text{fname,mint,lname}} (\sigma_{\text{gender=“F” and dname=“HR”}} (\text{Employee} * \text{Department}))$$

Which of the two alternatives is better?

```
SELECT fname, mint, lname
FROM Employee e JOIN Department d
ON e.dno = d.dno
WHERE e.gender = 'F'
  AND d.dname = 'HR';
```

RDBS performs query chain optimization to generate an optimal sequence of relational algebra statements.

Queries Using Set Operations, Projection, Conditional Selection and Joins

List the ids of all employees of the EEE department working on project Panopto.

ans

$$= \prod_{\text{empid}} (\text{Employee} * (\sigma_{\text{dname}=\text{“EEE”}}(\text{Department}) * \sigma_{\text{pname}=\text{“Panopto”}}(\text{Project})))$$

which attribute is being used to join?

```
SELECT DISTINCT e.empid
FROM

Employee e JOIN Department d
ON e.dno = d.dno

JOIN WorksOn w
ON e.empid = w.empid

JOIN Project p
ON w.pno = p.pno

WHERE d.dname = 'EEE'
  AND p.pname = 'Panopto';
```

Schema

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

List the last names of all employees who have at least one dependent.

$$\text{ans} = \prod_{\text{empid}} \text{Dependent}$$

```
SELECT empid FROM Dependent;
```

Get the last names of all employees who have at least two dependents.

???

Aggregation

- Syntax: **($_{\text{<Grouping attribute>}}$ F $_{\text{<function list>}}$ (relation name))**

- **<function list> :** contains simple mathematical functions.
    - Common functions are: MAX, MIN, AVG, SUM and COUNT
- **<Grouping attribute>:** contains a column name to organize the function outputs into groups.
    - Without a grouping attribute a global, across-the-table result is returned

Aggregation Example

$({}_{A} \mathcal{F}_{SUM(B),\, MAX(C)} (r))$

r	A	B	C
	1	10	1
	1	2	5
	2	3	3
	3	6	10
	3	5	7

	Group-by field		Summary Data
Temp	A	Sum_B	Max_C
	1	12	5
	2	3	3
	3	11	10

```
SELECT A,
      SUM(B) AS Sum_B,
      MAX(C) AS Max_C
FROM r
GROUP BY A;
```

Corresponding SQL

Renaming of column names required to have reusable column names, e.g., to be used in projection or conditional selection, etc.

Queries Using Aggregation

Get the employee ids and number of dependents of all employees who have at least two dependents.

```
SELECT e.empid,
    COUNT(*) AS num_dependents
FROM

Employee e JOIN Dependent d
  ON e.empid = d.empid

GROUP BY e.empid

HAVING COUNT(*) >= 2;
```

Schema

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

HAVING is used for conditional selection when **GROUP BY** based aggregation is involved.

Nested Queries (or Subqueries)

List the ids and names of employees who are working on some project.

```
SELECT empid, fname, mint, lname
FROM Employee
WHERE empid IN (
   SELECT empid
   FROM WorksOn
);
```

Schema

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

List the ids and names of employees who have at least one dependent.

```
SELECT e.empid, fname, mint, lname
FROM Employee e
WHERE EXISTS (
   SELECT *
   FROM Dependent d
   WHERE d.empid = e.empid
);
```

Other Subquery Operators like IN that can be used:

IN: value appears in the subquery result set
NOT IN: value does not appear in the subquery result set
EXISTS: subquery returns at least one row
NOT EXISTS: subquery returns no rows
ANY: comparison true for at least one value returned by subquery
SOME: same as ANY
ALL: comparison true for every value returned by subquery

Miscellaneous Queries

List the complete employee data of all employees, and if they are working on projects, display the total number of hours they spend across all their projects.

```
SELECT e.*, SUM(w.hours) AS total_hours

FROM Employee e LEFT JOIN WorksOn w
  ON e.empid = w.empid

GROUP BY e.empid;
```

Schema

Employee (fname, mint, lname, <u>empid</u>, bdate, address, gender, salary, superid, dno)

Department (dname, <u>dno</u>, mgrid, mgstartdate)

Project (pname, <u>pno</u>, plocation, dno)

WorksOn (<u>empid</u>, <u>pno</u>, hours)

Dependent(<u>empid</u>, <u>depname</u>, gender, <u>bdate</u>, relationship)

Miscellaneous Queries

List the ids and last names of all EEE employees working on maximum number of projects.

```
SELECT e.empid, e.lname
FROM Employee e
JOIN Department d ON e.dno = d.dno
JOIN WorksOn w   ON e.empid = w.empid
WHERE d.dname = 'EEE'
GROUP BY e.empid, e.lname
HAVING COUNT(*) = (
   SELECT MAX(ct)
   FROM (
      SELECT COUNT(*) AS ct
      FROM
      Employee e2 JOIN Department d2
      ON e2.dno = d2.dno
      JOIN WorksOn w2
      ON e2.empid = w2.empid
      WHERE d2.dname = 'EEE'
      GROUP BY e2.empid
   )
);
```

Schema

Employee (fname, mint, lname, empid, bdate, address, gender, salary, superid, dno)

Department (dname, dno, mgrid, mgstartdate)

Project (pname, pno, plocation, dno)

WorksOn (empid, pno, hours)

Dependent(empid, depname, gender, bdate, relationship)

Results in a single-column table with column name ct

Results in a single value.

Miscellaneous Queries

Output a table with the following columns:
Employee Id, Last Name, Dept. name, Salary, No. of Proj, No. of Dependents

```
SELECT
  e.empid AS "Employee Id",
  e.lname AS "Last Name",
  d.dname AS "Dept. name",
  e.salary AS "Salary",
  COUNT(DISTINCT w.pno) AS "No. of Proj",
  COUNT(DISTINCT dep.depname) AS "No. of Dependents"
FROM Employee e JOIN Department d
ON e.dno = d.dno
JOIN WorksOn w
ON e.empid = w.empid
JOIN Dependent dep
ON e.empid = dep.empid
GROUP BY
  e.empid, e.lname, d.dname, e.salary;
```

Schema

Employee (fname, mint, lname, empid, bdate, address, gender, salary, superid, dno)

Department (dname, dno, mgrid, mgstartdate)

Project (pname, pno, plocation, dno)

WorksOn (empid, pno, hours)

Dependent(empid, depname, gender, bdate, relationship)

RDBMS

A ***database*** is a structured collection of data that is managed by a DBMS (Database management system)

- Some of the most powerful and widely used database management systems are relational: **RDBMS**.
  - They provide a **high-level query language** called SQL, which is based on relational algebra.
  - Examples of RDBMS: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, **SQLite**, Amazon RDS, etc.

RDBMs Ensure ACID Properties

- **A**tomicity**:** transactions, which consist of multiple statements, are treated as atomic.
  - Either the whole thing succeeds or the whole thing fails
- **C**onsistency**:** transactions can only bring the database from one valid state to another
  - a state is valid if it follows all the specified rules including constraints
- **I**solation**:** ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially.
- **D**urability: database commits are final and durable even if system fails after a commit.

PART II
Practicing with SQL

- **Open**:
  https://sqliteonline.com/

- **Upload Schema/Data**:
  – Download **emp_db.sql**
  – Import into https://sqliteonline.com/

Write SQL for the Following Queries

1) List the first name, last name, and salary of all employees whose salary is at least 80000.

2) List the employee id and department number of all employees whose gender is 'F' and who live at an address containing 'London' (use LIKE '%London%')

3) List the department name and manager start date for all departments whose department number is greater than 20.

Expected result:

dname	mgstartdate
Finance	2021-04-10
Sales	2020-06-20
IT	2024-02-01

4) List the employee ids and last names of all employees who work in the HR department.

Expected result:

empid	lname
1004	Baker
1005	Patel

5) List the project names and locations for all projects controlled by the Engineering department.

Expected result:

pname	plocation
Product Alpha	Cambridge

continued…

6) List the employee ids and project numbers for all employees who work more than 15 hours on a single project.

Expected result:

empid	pno
1008	2002
1001	2004
1003	2004

7) For each department, list the department name and the number of employees in that department.

Expected result:

dname	num_employees
Engineering	3
Finance	1
HR	2
IT	1
Sales	1

continued…

8) List the employee ids and last names of employees who work on at least two projects.

Expected result:

empid	lname
1008	Ali

9) List the employee ids and last names of employees whose salary is higher than the average salary in their own department.

Expected result:

empid	lname
1001	Rahman
1004	Baker

10) List the employee ids and last names of employees who work on the maximum number of projects in the entire company.

Expected result:

empid	lname
1008	Ali

More Queries to Attempt

11. *List the first and last names of all employees along with their department names.*
12. *List the project names and the last names of employees working on each project.*
13. *List the employee ids and total number of hours each employee works across all projects.*
14. *List the department names and the total salary paid to employees in each department.*
15. *List the ids and last names of employees who have no dependents.*
16. *List the project names of projects that currently have no employees assigned.*
17. *List the ids and last names of employees who work on every project.*
18. *List the department names that have more than three employees.*
19. *List the ids and last names of employees who earn the highest salary in their department.*
20. *List the ids and last names of employees whose total project hours exceed the company-wide average total project hours per employee.*