Multiple Tables

- queries often need data from more than one table

List employees earning more than $70,000 – include the employee's name, sex, salary, and the name of the department they work for.

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

fname	minit	lname	ssn	bdate	address	sex	salary	superssn	deptnum
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

deptname	deptnum	mgrssn	mgrstartdate
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

deptnum	deptloc
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

empssn	projnum	hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0

PROJECT

projname	projnum	projloc	deptnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4

CROSS JOIN and Cartesian Product

- *R* `CROSS JOIN` *S*
- *R*,*S* (Cartesian product)

equivalent syntax
`CROSS JOIN` is preferred because it makes the operation explicit

  - generates all pairs of rows from R and S
    - result table has every column from both R and S with one row for each possible pairing of a row from R and a row from S

Table: `SIZES`

size_id	size_name
1	Small
2	Medium

Table: `COLORS`

color_id	color_name
10	Red
20	Blue
30	Green

Result (2 × 3 = 6 rows):

size_id	size_name	color_id	color_name
1	Small	10	Red
1	Small	20	Blue
1	Small	30	Green
2	Medium	10	Red
2	Medium	20	Blue
2	Medium	30	Green

```
SELECT *
FROM SIZES CROSS JOIN COLORS
```

```
SELECT *
FROM SIZES, COLORS
```


Examples

```
SELECT deptname, deptloc
FROM DEPARTMENT CROSS JOIN DEPT_LOCATIONS
```

all pairs of department and department location

```
SELECT D1.deptname, D2.deptname
FROM DEPARTMENT D1 CROSS JOIN DEPARTMENT D2
WHERE D1.mgrstartdate < D2.mgrstartdate
```

pairs of departments where the first department's manager started before the second

```
SELECT D1.deptname, D2.deptname
FROM DEPARTMENT D1 CROSS JOIN DEPARTMENT D2
WHERE D1.mgrstartdate <= D2.mgrstartdate AND D1.deptnum <> D2.deptnum
```

pairs of distinct departments where the first department's manager started no later the second

Inner Joins

- *R* `[INNER] JOIN` *S* `ON` *C*
  - rows with a match on both sides, where "match" is determined by C
    - result table has every column from both R and S with only the pairings of a row from R and a row from S for which C is true
  - equivalent to `CROSS JOIN` with `WHERE`

Table: `CUSTOMERS`

customer_id	customer_name
1	Alice
2	Bob
3	Carol
4	Dave

Table: `ORDERS`

order_id	customer_id	product
101	1	Laptop
102	1	Mouse
103	2	Keyboard
104	5	Monitor

INNER JOIN — only rows with a match on both sides

customer_id	customer_name	order_id	customer_id	product
1	Alice	101	1	Laptop
1	Alice	102	1	Mouse
2	Bob	103	2	Keyboard

```
SELECT *
FROM CUSTOMERS JOIN ORDERS
 ON CUSTOMERS.customer_id=ORDERS.customer_id
```

```
SELECT *
FROM CUSTOMERS CROSS JOIN ORDERS
WHERE CUSTOMERS.customer_id=ORDERS.customer_id
```

Examples

```
SELECT fname, lname, deptname
FROM EMPLOYEE JOIN DEPARTMENT
 ON EMPLOYEE.deptnum = DEPARTMENT.deptnum
```

employees (name) and the name of the department they work in

```
SELECT E1.fname, E1.lname, E2.fname, E2.lname
FROM EMPLOYEE E1 JOIN EMPLOYEE E2
 ON E1.superssn = E2.ssn AND E1.deptnum = E2.deptnum
```

pairs of employees where the first employee's supervisor is the second employee and both are in the same department

```
SELECT EMPLOYEE.fname, EMPLOYEE.lname, PROJECT.projname, WORKS_ON.hours
FROM EMPLOYEE
 JOIN WORKS_ON ON EMPLOYEE.ssn = WORKS_ON.empssn
 JOIN PROJECT ON WORKS_ON.projnum = PROJECT.projnum
```

work assignments and hours worked with employee and project names

```
SELECT EMPLOYEE.fname, EMPLOYEE.lname, PROJECT.projname, WORKS_ON.hours
FROM EMPLOYEE
 JOIN WORKS_ON ON EMPLOYEE.ssn = WORKS_ON.empssn
 JOIN PROJECT ON WORKS_ON.projnum = PROJECT.projnum
  AND EMPLOYEE.deptnum = PROJECT.deptnum
```

work assignments and hours worked with employee and project names, for assignments where the employee and project are in the same department

Examples

```
SELECT deptname, fname, lname
FROM DEPARTMENT
 JOIN EMPLOYEE ON DEPARTMENT.deptnum = EMPLOYEE.deptnum
  OR DEPARTMENT.mgrssn = EMPLOYEE.ssn
```

departments together with any employee who either works in the department or manages it

```
SELECT E1.fname, E1.lname, E2.fname, E2.lname
FROM EMPLOYEE E1 JOIN EMPLOYEE E2
 ON E1.salary < E2.salary
```

pairs of employees where the first employee's salary is less than the second's

```
SELECT E1.fname, E1.lname, E2.fname, E2.lname
FROM EMPLOYEE E1 JOIN EMPLOYEE E2
 ON E1.deptnum = E2.deptnum AND E1.bdate > E2.bdate
```

pairs of employees in the same department where the first employee's birthdate is after the second's

`USING` and Natural Joins

- *R* `JOIN` *S* `USING (`*a1*`,`*a2*`,…)`
  - rows with a match on both sides, where "match" is determined by the named columns
    - equivalent to `R JOIN S ON C` where C is the conjunction of equality conditions on the named columns (`R.a1=S.a1 AND R.a2=S.a2 AND …`)
  - result has only one copy of each join column
- *R* `NATURAL JOIN` *S*
  - equivalent to USING with all of the columns with the same name in both R and S

Table: `CUSTOMERS`

customer_id	customer_name
1	Alice
2	Bob
3	Carol
4	Dave

Table: `ORDERS`

order_id	customer_id	product
101	1	Laptop
102	1	Mouse
103	2	Keyboard
104	5	Monitor

customer_id	customer_name	order_id	product
1	Alice	101	Laptop
1	Alice	102	Mouse
2	Bob	103	Keyboard

```
SELECT *
FROM CUSTOMERS JOIN ORDERS USING (customer_id)
```

```
SELECT *
FROM CUSTOMERS JOIN ORDERS
 ON CUSTOMERS.customer_id=ORDERS.customer_id
```

```
SELECT *
FROM CUSTOMERS NATURAL JOIN ORDERS
```


Examples

```
SELECT deptname, deptnum, deptloc
FROM DEPARTMENT JOIN DEPT_LOCATIONS USING (deptnum)
```

departments and their locations

```
SELECT fname, lname, deptname, deptloc
FROM EMPLOYEE JOIN DEPARTMENT USING (deptnum)
 JOIN DEPT_LOCATIONS USING (deptnum)
```

employees, their department, and its locations
(each employee is listed once for each of their department's locations)

```
SELECT projname, projnum, empssn, hours
FROM PROJECT NATURAL JOIN WORKS_ON
```

work assignments, with the project name
(equivalent to `PROJECT JOIN WORKS_ON USING (projnum)`)

```
SELECT projname, projnum, fname, lname, hours
FROM PROJECT NATURAL JOIN WORKS_ON NATURAL JOIN EMPLOYEE
```

note that this is *not* work assignments with the project name and the employee's name – it actually combines employees with projects in their department, not with their work assignments
(equivalent to `PROJECT JOIN WORKS_ON USING (projnum) JOIN EMPLOYEE USING (deptnum)`)

```
SELECT EMPLOYEE.fname, EMPLOYEE.lname, DEPENDENT.relationship
FROM EMPLOYEE NATURAL JOIN DEPENDENT
```

note that this is *not* employees with the relationships of their dependents – it's actually pairs of employees with dependents having the same first name, sex, and birthdate
(equivalent to `EMPLOYEE JOIN DEPENDENT USING (fname,sex,birthdate)`)

Outer Joins

Inner joins combine matching rows in two relations. But what if a row in one relation has no match in the other?

An *outer join* includes rows with no match in the other relation, filling in the missing columns with NULL.

- `R LEFT [OUTER] JOIN S ON C` includes all rows in R, including those with no match in S
- `R RIGHT [OUTER] JOIN S ON C` includes all rows in S, including those with no match in R
- `R FULL [OUTER] JOIN S ON C` includes all rows in R and in S, including those with no match in the other table
  - full outer joins are not supported by MySQL

- `OUTER JOIN`s can be `NATURAL` instead of including an `ON` condition.
- `ON` and `WHERE` conditions are *not* necessarily interchangeable for outer joins

Outer Joins

Table: `CUSTOMERS`

customer_id	customer_name
1	Alice
2	Bob
3	Carol
4	Dave

Table: `ORDERS`

order_id	customer_id	product
101	1	Laptop
102	1	Mouse
103	2	Keyboard
104	5	Monitor

LEFT JOIN — all customers, matched orders where available

customer_id	customer_name	order_id	customer_id	product
1	Alice	101	1	Laptop
1	Alice	102	1	Mouse
2	Bob	103	2	Keyboard
3	Carol	NULL	NULL	NULL
4	Dave	NULL	NULL	NULL

RIGHT JOIN — all orders, matched customers where available

customer_id	customer_name	order_id	customer_id	product
1	Alice	101	1	Laptop
1	Alice	102	1	Mouse
2	Bob	103	2	Keyboard
NULL	NULL	104	5	Monitor

FULL OUTER JOIN — everything from both sides, matched where possible

customer_id	customer_name	order_id	customer_id	product
1	Alice	101	1	Laptop
1	Alice	102	1	Mouse
2	Bob	103	2	Keyboard
3	Carol	NULL	NULL	NULL
4	Dave	NULL	NULL	NULL
NULL	NULL	104	5	Monitor

Examples

```
SELECT deptname, projname
FROM DEPARTMENT LEFT JOIN PROJECT
ON DEPARTMENT.deptnum = PROJECT.deptnum
```

departments with their projects, if any
(includes every department, even if it has no projects)

```
SELECT deptname, projname
FROM PROJECT RIGHT JOIN DEPARTMENT
ON DEPARTMENT.deptnum = PROJECT.deptnum
```

departments with their projects, if any
(equivalent to `DEPARTMENT LEFT JOIN PROJECT`)

```
SELECT deptname
FROM DEPARTMENT LEFT JOIN PROJECT ON DEPARTMENT.deptnum = PROJECT.deptnum
WHERE PROJECT.projnum IS NULL
```

departments with no projects

Examples

```
SELECT deptname, projname
FROM DEPARTMENT LEFT JOIN PROJECT
 ON DEPARTMENT.deptnum = PROJECT.deptnum AND PROJECT.projloc = 'Houston'
```

departments with their Houston projects
(includes every department, even if it has no projects in Houston)

```
SELECT deptname, projname
FROM DEPARTMENT LEFT JOIN PROJECT ON DEPARTMENT.deptnum = PROJECT.deptnum
WHERE PROJECT.projloc = 'Houston'
```

departments with projects in Houston
(excludes departments without projects in Houston – the `WHERE` is applied to the result of the `JOIN`)

```
SELECT EMPLOYEE.fname, EMPLOYEE.lname, PROJECT.projname
FROM EMPLOYEE LEFT JOIN PROJECT
 ON EMPLOYEE.deptnum = PROJECT.deptnum AND EMPLOYEE.salary > 80000
```

employees, with those earning over $80,000 paired with any projects in their department
(all employees are listed – to include only employees earning over $80,000, move `EMPLOYEE.salary > 80000` to `WHERE`)

Aliases

- must qualify column name with relation name or alias to disambiguate same-named columns

```
SELECT fname, lname, deptname
FROM EMPLOYEE JOIN DEPARTMENT
 ON EMPLOYEE.deptnum = DEPARTMENT.deptnum

SELECT fname, lname, deptname
FROM EMPLOYEE E JOIN DEPARTMENT D
 ON E.deptnum = D.deptnum

SELECT E1.fname, E1.lname, E2.fname, E2.lname
FROM EMPLOYEE E1 JOIN EMPLOYEE E2
 ON E1.superssn = E2.ssn AND E1.deptnum = E2.deptnum
```

  - using the first letter of the table name is common and convenient when there is no ambiguity and queries are not too complex
  - meaningful abbreviations (e.g. `EMP`, `DEPT`) are preferred when queries involve many tables and across shared codebases
  - use aliases consistently across a codebase
    - e.g. always use the same alias for a particular table across queries

Anti-Joins

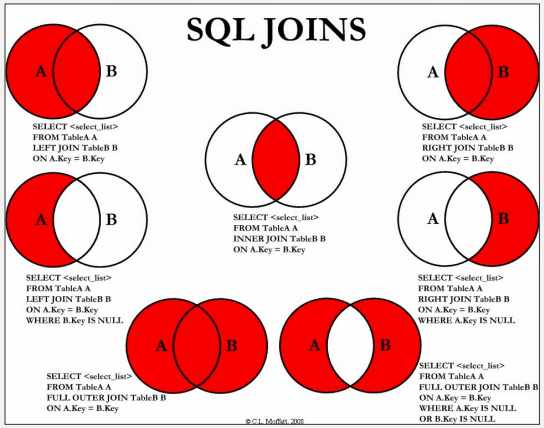
- *anti-join* is a pattern that returns rows from one table that have no matching row in another
  - use a `LEFT JOIN`, then pick rows where a right side column has `NULL` values

```
SELECT fname, lname
FROM EMPLOYEE LEFT JOIN DEPENDENT ON EMPLOYEE.ssn = DEPENDENT.empssn
WHERE DEPENDENT.empssn IS NULL
```


Best Practices

- for inner joins –
  - use `ON` only for true join conditions
    - "is this condition (part of) establishing a meaningful relationship between the tables?" – typically reflect foreign keys
  - prefer `JOIN` for equality conditions (equijoins) and `CROSS JOIN` with `WHERE` for non-equality condition
  - prefer `JOIN` for conjunctions (`AND`) and `UNION` for disjunctions (`OR`)
- syntax alternatives
  - prefer `CROSS JOIN` to `,` (Cartesian product)
    - makes the join operation explicit
  - prefer `JOIN … ON` or `JOIN … USING` to `NATURAL JOIN`
    - makes the join logic explicit
    - avoids bugs due to coincidentally same-named columns
    - avoids silent behavior changes when the schema changes
  - prefer `LEFT JOIN` to `RIGHT JOIN`
    - every `RIGHT JOIN` can be rewritten as a `LEFT JOIN`
    - `LEFT JOIN` is a more natural reading order

Join Summary



 https://javarevisited.blogspot.com/2012/11/how-to-join-three-tables-in-sql-query-mysql-sqlserver.html#