Outer Joins

- Include in join result non-matching tuples
- Result tuple padded with NULL values
- Variants
  - FULL: non-matching tuples in both relations included in result
  - LEFT: only non-matching tuples in left relation included in result
  - RIGHT: only non-matching tuples in right relation included in result

The explicit syntax for an outer join
SELECT select_list
FROM table_1
    {LEFT|RIGHT|FULL} [OUTER] JOIN table_2 ...

What outer joins do
- Join
- Keeps unmatched rows from
- Left: The left table
- Right: The right table
- Full: Both tables

Outer Joins

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>

SELECT sid, sname, rating, age, bid, day
FROM Sailors NATURAL LEFT OUTER JOIN Reserves

Joins with a nullable FK in use
emp(eid, ename, age, salary)
dep(did, dname, budget, managerid)
nullable, FK to emp

Suppose there’s a new department without a manager--
SQL> select * from dept;

<table>
<thead>
<tr>
<th>DID</th>
<th>DNAME</th>
<th>BUDGET</th>
<th>MANAGERID</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>newone</td>
<td>1000000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Hardware</td>
<td>1048572.12</td>
<td>141582651</td>
</tr>
<tr>
<td>2</td>
<td>Operations</td>
<td>4099101</td>
<td>287321212</td>
</tr>
<tr>
<td>3</td>
<td>Legal</td>
<td>222988.13</td>
<td>248985255</td>
</tr>
<tr>
<td>4</td>
<td>Marketing</td>
<td>538099.54</td>
<td>548977562</td>
</tr>
<tr>
<td>5</td>
<td>Software</td>
<td>400011.12</td>
<td>141582651</td>
</tr>
<tr>
<td>6</td>
<td>Production</td>
<td>12099101</td>
<td>578875478</td>
</tr>
<tr>
<td>7</td>
<td>Shipping</td>
<td>5</td>
<td>489456522</td>
</tr>
</tbody>
</table>

8 rows selected.

Suppose there’s a new department without a manager--
SQL> select * from dept;

<table>
<thead>
<tr>
<th>DID</th>
<th>DNAME</th>
<th>BUDGET</th>
<th>MANAGERID</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>newone</td>
<td>1000000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Hardware</td>
<td>1048572.12</td>
<td>141582651</td>
</tr>
<tr>
<td>2</td>
<td>Operations</td>
<td>4099101</td>
<td>287321212</td>
</tr>
<tr>
<td>3</td>
<td>Legal</td>
<td>222988.13</td>
<td>248985255</td>
</tr>
<tr>
<td>4</td>
<td>Marketing</td>
<td>538099.54</td>
<td>548977562</td>
</tr>
<tr>
<td>5</td>
<td>Software</td>
<td>400011.12</td>
<td>141582651</td>
</tr>
<tr>
<td>6</td>
<td>Production</td>
<td>12099101</td>
<td>578875478</td>
</tr>
<tr>
<td>7</td>
<td>Shipping</td>
<td>5</td>
<td>489456522</td>
</tr>
</tbody>
</table>

8 rows selected.

Suppose there’s a new department without a manager--
SQL> select * from dept;

<table>
<thead>
<tr>
<th>DID</th>
<th>DNAME</th>
<th>BUDGET</th>
<th>MANAGERID</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>newone</td>
<td>1000000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Hardware</td>
<td>1048572.12</td>
<td>141582651</td>
</tr>
<tr>
<td>2</td>
<td>Operations</td>
<td>4099101</td>
<td>287321212</td>
</tr>
<tr>
<td>3</td>
<td>Legal</td>
<td>222988.13</td>
<td>248985255</td>
</tr>
<tr>
<td>4</td>
<td>Marketing</td>
<td>538099.54</td>
<td>548977562</td>
</tr>
<tr>
<td>5</td>
<td>Software</td>
<td>400011.12</td>
<td>141582651</td>
</tr>
<tr>
<td>6</td>
<td>Production</td>
<td>12099101</td>
<td>578875478</td>
</tr>
<tr>
<td>7</td>
<td>Shipping</td>
<td>5</td>
<td>489456522</td>
</tr>
</tbody>
</table>

8 rows selected.

Find department names and their manager names--
SQL> Select d.dname, e.ename from dept d, emp e
where d.managerid = e.eid;

<table>
<thead>
<tr>
<th>DNAME</th>
<th>ENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>Mary Johnson</td>
</tr>
<tr>
<td>Hardware</td>
<td>Mary Johnson</td>
</tr>
<tr>
<td>Legal</td>
<td>Barbara Wilson</td>
</tr>
<tr>
<td>Operations</td>
<td>Michael Miller</td>
</tr>
<tr>
<td>Shipping</td>
<td>Linda Davis</td>
</tr>
<tr>
<td>Marketing</td>
<td>Donald King</td>
</tr>
<tr>
<td>Production</td>
<td>Edward Baker</td>
</tr>
</tbody>
</table>

7 rows selected.

What happened to "newone"?! WHERE null = e.eid: unknown, excluded
Outer join to the rescue!

"Find department names and their manager names"

SQL> Select d.dname, e.ename from dept d LEFT OUTER JOIN emp e

<table>
<thead>
<tr>
<th>DNAME</th>
<th>ENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Software</td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
</tr>
<tr>
<td></td>
<td>Shipping</td>
</tr>
<tr>
<td></td>
<td>Legal</td>
</tr>
<tr>
<td></td>
<td>Marketing</td>
</tr>
<tr>
<td></td>
<td>Production</td>
</tr>
<tr>
<td></td>
<td>newone</td>
</tr>
</tbody>
</table>

8 rows selected.

Remember, if you join using a nullable FK, you may want an outer join.

The Departments table

<table>
<thead>
<tr>
<th>DEPT_NAME</th>
<th>DEPT_NO</th>
<th>LAST_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>1</td>
<td>Hernandez</td>
</tr>
<tr>
<td>Maintenance</td>
<td>5</td>
<td>Hardy</td>
</tr>
<tr>
<td>Operations</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Payroll</td>
<td>2</td>
<td>Aaronson</td>
</tr>
<tr>
<td>Payroll</td>
<td>2</td>
<td>Simonian</td>
</tr>
<tr>
<td>Personnel</td>
<td>4</td>
<td>Jones</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Locario</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Watson</td>
</tr>
</tbody>
</table>

A SELECT statement that uses a left outer join

SELECT vendor_name, invoice_number, invoice_total
FROM vendors LEFT JOIN invoices
ON vendors.vendor_id = invoices.vendor_id
ORDER BY vendor_name

The result set

<table>
<thead>
<tr>
<th>CHECK_RUN</th>
<th>INVOICE_NUM</th>
<th>INVOICE_TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>mail</td>
<td>mail</td>
</tr>
<tr>
<td>Sales</td>
<td>Office</td>
<td>mail</td>
</tr>
<tr>
<td>Marketing</td>
<td>Office</td>
<td>mail</td>
</tr>
<tr>
<td>Marketing</td>
<td>Office</td>
<td>mail</td>
</tr>
</tbody>
</table>

(202 rows selected)

This shows that vendor “ASC Signs” is in the system but has no invoices currently, etc.

A left outer join

SELECT department_name AS dept_name,
    d.department_number AS dept_no,
    last_name
FROM departments d
    LEFT JOIN employees e
    ON d.department_number = e.department_number
ORDER BY department_name

• This join preserves departments even if they have no employees
• It would still be useful even if we had a non-null FK in place.

DEPT_NAME | DEPT_NO | LAST_NAME |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>1</td>
<td>Hernandez</td>
</tr>
<tr>
<td>Maintenance</td>
<td>5</td>
<td>Hardy</td>
</tr>
<tr>
<td>Operations</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Payroll</td>
<td>2</td>
<td>Aaronson</td>
</tr>
<tr>
<td>Payroll</td>
<td>2</td>
<td>Simonian</td>
</tr>
<tr>
<td>Personnel</td>
<td>4</td>
<td>Jones</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Locario</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Watson</td>
</tr>
</tbody>
</table>

A full outer join

SELECT department_name AS dept_name,
    d.department_number AS d_dept_no,
    e.department_number AS e_dept_no,
    last_name
FROM departments d
    FULL JOIN employees e
    ON d.department_number = e.department_number
ORDER BY department_name

• This join preserves employees even if they have non-matching departments (i.e. have violations of the proposed FK)
• If we had a non-null FK in place, we would simply use the inner join.

DEPT_NAME | DEPT_NO | LAST_NAME |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>1</td>
<td>Hernandez</td>
</tr>
<tr>
<td>Maintenance</td>
<td>5</td>
<td>Hardy</td>
</tr>
<tr>
<td>Operations</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Payroll</td>
<td>2</td>
<td>Aaronson</td>
</tr>
<tr>
<td>Payroll</td>
<td>2</td>
<td>Simonian</td>
</tr>
<tr>
<td>Personnel</td>
<td>4</td>
<td>Jones</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Locario</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Watson</td>
</tr>
</tbody>
</table>
• A right join can be converted to a left join by switching the order of the tables.

• Murach says practitioners prefer left joins (pg. 136)

A SELECT statement that uses left outer joins

```sql
SELECT department_name, last_name, project_number AS proj_no
FROM departments d
LEFT JOIN employees e
ON d.department_number = e.department_number
LEFT JOIN projects p
ON e.employee_id = p.employee_id
ORDER BY department_name, last_name, project_number
```

A SELECT statement that uses full outer joins

```sql
SELECT department_name, last_name, project_number AS proj_no
FROM departments dpt
FULL JOIN employees emp
ON dpt.department_number = emp.department_number
FULL JOIN projects prj
ON emp.employee_id = prj.employee_id
ORDER BY department_name
```

The implicit syntax for an outer join (Oracle)

```sql
SELECT select_list
FROM table_1, table_2, table_3...
WHERE table_1.column_name = table_2.column_name [+] [table_2.column_name = table_3.column_name [+]...
```

FYI: Oracle offered outer join before the standardization, using this awkward syntax. We won’t use it.

A SELECT statement with an outer and inner join

```sql
SELECT department_name AS dept_name, last_name, project_number
FROM departments dpt
LEFT JOIN employees emp
ON dpt.department_number = emp.department_number
LEFT JOIN projects prj
ON emp.employee_id = prj.employee_id
ORDER BY department_name
```

The result set

```
<table>
<thead>
<tr>
<th>department_name</th>
<th>last_name</th>
<th>project_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>Alexander</td>
<td>P01</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Barry</td>
<td>P01</td>
</tr>
<tr>
<td>Operations</td>
<td>Bill</td>
<td>P01</td>
</tr>
<tr>
<td>Payroll</td>
<td>Jessica</td>
<td>P01</td>
</tr>
<tr>
<td>Personal</td>
<td>John</td>
<td>P01</td>
</tr>
<tr>
<td>Engineering</td>
<td>Karen</td>
<td>P01</td>
</tr>
<tr>
<td>Sales</td>
<td>Larry</td>
<td>P01</td>
</tr>
<tr>
<td>Sales</td>
<td>Mary</td>
<td>P01</td>
</tr>
<tr>
<td>Sales</td>
<td>O’Larry</td>
<td>P01</td>
</tr>
</tbody>
</table>
```

(7 rows selected)

Here we’re saying we are only interested in employees in known departments, but want to include employees with no projects.
The syntax for a join with the USING keyword

```
SELECT select_list
FROM table_1
    [LEFT|RIGHT|FULL] JOIN table_2
USING(junoin_column_1[, join_column_2]...)
```

A SELECT statement with the USING keyword

```
SELECT invoice_number, vendor_name
FROM invoices
JOIN vendors USING (vendor_id)
ORDER BY invoice_number
```

The result set

```
<table>
<thead>
<tr>
<th>INVOICE_NUMBER</th>
<th>VENDOR_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1013</td>
<td>Mailco, Inc.</td>
</tr>
<tr>
<td>1094</td>
<td>Mailco, Inc.</td>
</tr>
<tr>
<td>7313</td>
<td>Mailco, Inc.</td>
</tr>
<tr>
<td>73141</td>
<td>Mailco, Inc.</td>
</tr>
</tbody>
</table>
```

(114 rows selected)

Integrity Constraints (Review)

- An IC describes conditions that every legal instance of a relation must satisfy.
- Inserts/deletes/updates that violate IC’s are disallowed.
- Types of IC’s:
  - domain constraints
    - Field values must be of right type - always enforced
  - primary key constraints
  - foreign key constraints
  - general constraints

Enforcing Referential Integrity

- What should be done if an Enrolled tuple with a non-existent student id is inserted?
  - Reject the insert!
- What should be done if a Students tuple is deleted?
  - Delete all Enrolled tuples that refer to it
  - Correct as far as IC is concerned, but data is lost!
  - Disallow deletion of a Students tuple that is referred to
  - More appropriate in practice
  - Set sid in Enrolled tuples that refer to it to a default sid
  - Or, set it to NULL but this is not appropriate for a relationship table like enrolled.
Referential Integrity in SQL

SQL/92 and SQL:1999 support all options on deletes and updates.
- Default is **NO ACTION** (delete/update is rejected)
- **CASCADE** (delete/update all tuples that refer to deleted/updated tuple)
- **SET NULL** / **SET DEFAULT** (sets foreign key value of referencing tuple)

CREATE TABLE Enrolled
(sid CHAR(20) default '00',
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
REFERENCES Students (sid)
on delete set default on update cascade)

More on this example...

CREATE TABLE Enrolled
(sid CHAR(20) default '00',
cid CHAR(20), grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid) REFERENCES Students (sid)
on delete set default on update cascade)

This case is discussed on pg. 71. If student '123' is deleted from Students, this would replace that student's enrolled rows to ('00', cid), making it appear that student '00' was enrolled in whatever course(s) student '123' had been in. This approach is properly rejected.

The better solution is **ON DELETE CASCADE** or the default **NO ACTION**, depending on whether you need to keep old enrollment information for deleted students.

Complex Constraints: **CHECK** clause

- Useful when more general ICs than keys are involved
- Can use logical expression to express constraint
- Constraints can be named
- Not checked if table is empty!
- Available in Oracle, SQL Server, DB2... (in Entry SQL-92)
- MySQL: parses it but ignores it.

CREATE TABLE Sailors
(sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CONSTRAINT RatingRange
CHECK (rating >= 1
AND rating <= 10 )
)

(Like example on pg. 165)

Check clause: second text example, pg. 166

CREATE TABLE Reserves
(sid INTEGER,
bid INTEGER,
day DATE,
FOREIGN KEY (sid) REFERENCES Sailors,
FOREIGN KEY (bid) REFERENCES Boats,
CONSTRAINT noInterlakeRes
CHECK ( 'Interlake' <> (SELECT B.bname
FROM Boats B
WHERE B.bid = Reserves.bid ))
)

Subquery in check condition: allowed only in Full SQL-92, so not portable. Oracle: ERROR at line 8:
ORA-02251: subquery not allowed here

So practical rule: **CHECK** can access this table only if it is local to the table; i.e., the table for Sailors is visible to the table for Reserves.

The syntax of a check constraint

```sql
[CONSTRAINT constraint_name] CHECK (condition)
```

A statement with column check constraints

```sql
CREATE TABLE invoices
( invoice_id NUMBER          PRIMARY KEY,  
invoice_total NUMBER(9,2)     NOT NULL,  
payment_total NUMBER(9,2)     DEFAULT 0,  
CHECK (invoice_total >= 0),  
CHECK (payment_total >= 0)  
)
```

This syntax is not covered in R&G, but is portable.

A statement with table-level check constraints

```sql
CREATE TABLE invoices
( invoice_id NUMBER PRIMARY KEY,  
invoice_total NUMBER(9,2) NOT NULL,  
payment_total NUMBER(9,2) DEFAULT 0,  
CONSTRAINT invoices_ck CHECK (invoice_total >= 0  
AND payment_total >= 0)  
)
```

This fits the R&G book's syntax, i.e., R&G only covers table-level check constraints.
An INSERT statement that fails due to a check constraint

```
INSERT INTO invoices
VALUES (1, 99.99, -10)
```

The response from the system

```
SQL Error: ORA-02290: check constraint (EX.INVOICES_CK) violated 02290. 00000 = "check constraint (%s.%s) violated"
```

*Cause:* The values being inserted do not satisfy the named check

*Action:* do not insert values that violate the constraint.

---

Complex Constraints: Assertions

- *Number of boats plus number of sailors is < 100*
- Not associated with a particular table
- Constraint may apply to multiple tables!
- But only in Full SQL-92, so not portable
- Not supported by Oracle, mysql, ...
- We won’t use this feature.

```
CREATE ASSERTION smallClub
CHECK
  (SELECT COUNT (S.sid) FROM Sailors S)
  + (SELECT COUNT (B.bid) FROM Boats B) < 100
```