Outerjoin

$R \overset{\circ}{\bowtie} S = R \bowtie S$ with *dangling* tuples padded with nulls and included in the result.

- A tuple is dangling if it doesn't join with any other tuple.

$R =$

A	B
1	2
3	4

$S =$

B	C
2	5
2	6
7	8

$R \overset{\circ}{\bowtie} S =$

A	B	C
1	2	5
1	2	6
3	4	NULL
NULL	7	8

Outerjoin in SQL2

A number of forms are provided.

- Can be used either stand-alone (in place of a select-from-where) or to define a relation in the `FROM`-clause.

  R `NATURAL JOIN` S
  R `JOIN` S `ON` condition
    e.g., condition: $R.B = S.B$
  R `CROSS JOIN` S
  R `OUTER JOIN` S

- The latter can be modified by:

  1. Optional `NATURAL` in front of `JOIN`.
  2. Optional `ON` condition at end.
  3. Optional `LEFT`, `RIGHT`, or `FULL` before `OUTER`.

  ✦ `LEFT` = pad dangling tuples of R only; `RIGHT` = pad dangling tuples of S only.

Oracle Outerjoin

Ain't no such thing.

- But parenthesized select-from-where allowed in a `FROM` clause.
    - Really a way to define a view and use it in a single query.

Example

Find the average over all bars of the maximum price the bar charges for a beer.

```
Sells(bar, beer, price)

SELECT AVG(maxPrice)
FROM (SELECT bar, MAX(price)
          AS maxPrice
         FROM Sells
         GROUP BY Bar);
```

Problem

Can we express the outerjoin in Oracle SQL as some more complicated expression?

Constraints

Commercial relational systems allow much more "fine-tuning" of constraints than do the modeling languages we learned earlier.

- In essence: SQL programming is used to describe constraints.

Outline

1. Primary key declarations (covered).
2. Foreign-keys = referential integrity constraints.
    - E.g., if `Sells` mentions a beer, then we should be able to find that beer in `Beers`.
3. Attribute- and tuple-based checks = constraints within relations.
4. SQL2 Assertions = global constraints.
    - Not found in Oracle 7.3.2.
5. Oracle Triggers.
    - A substitute for assertions.
6. SQL3 triggers and assertions.

Foreign Keys

In relation R a clause that "attribute A references $S(B)$" says that whatever values appear in the A column of R must also appear in the B column of relation S.

- B must be declared the primary key for S.

Example

```
CREATE TABLE Beers (
    name CHAR(20) PRIMARY KEY,
    manf CHAR(20)
);

CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(20) REFERENCES
            Beers(name),
    price REAL
);
```

- Alternative: add another element declaring the foreign key, as:

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(20),
    price REAL,
    FOREIGN KEY beer REFERENCES
            Beers(name)
);
```

- Extra element essential if the foreign key is more than one attribute.

What Happens When a Foreign Key Constraint is Violated?

- Two ways:

1. Insert a `Sells` tuple referring to a nonexistent beer.
   - ✦ Always rejected.
2. Delete or update a `Beers` tuple that has a `beer` value some `Sells` tuples refer to.
   a) Default: reject.
   b) *Cascade*: Ripple changes to referring `Sells` tuple.

Example

- Delete "Bud." Cascade deletes all `Sells` tuples that mention Bud.
- Update "Bud" → "Budweiser." Change all `Sells` tuples with "Bud" in `beer` column to be "Budweiser."

c) *Set Null*: Change referring tuples to have `NULL` in referring components.

Example

- Delete "Bud." Set-null makes all `Sells` tuples with "Bud" in the `beer` component have `NULL` there.
- Update "Bud" $\rightarrow$ "Budweiser." Same change.

Selecting a Policy

Add `ON [DELETE, UPDATE] [CASCADE, SET NULL]` to declaration of foreign key.

Example

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(20),
    price REAL,
    FOREIGN KEY beer REFERENCES
        Beers(name)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);
```

- "Correct" policy is a design decision.
  - E.g., what does it mean if a beer goes away? What if a beer changes its name?

Attribute-Based Checks

Follow an attribute by a condition that must hold for that attribute in each tuple of its relation.

- Form: `CHECK` (condition).
    - Condition may involve the checked attribute.
    - Other attributes and relations may be involved, but *only* in subqueries.
    - Oracle 7.3.2: *No subqueries allowed in condition.*
- Condition is checked only when the associated attribute changes (i.e., an insert or update occurs).

Example

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(20) CHECK(
        beer IN (SELECT name
            FROM Beers)
    ),
    price REAL CHECK(
        price <= 5.00
    )
);
```

- Check on `beer` is like a foreign-key constraint, except:
    - The check occurs only when we add a tuple or change the beer in an existing tuple, not when we delete a tuple from `Beers`.

Tuple-Based Checks

Separate element of table declaration.

- Form: like attribute-based check.
- But condition can refer to any attribute of the relation.
    - ✦ Or to other relations/attributes in subqueries.
    - ✦ Again: Oracle 7.3.2 forbids the use of subqueries.

Example

Only Joe's Bar can sell beer for more than \$5.

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(20),
    price REAL,
    CHECK(bar = 'Joe''s Bar' OR
        price <= 5.00)
);
```

Triggers

Often called event-condition-action rules.

- *Event* = a class of changes in the DB, e.g., "insert into `Beers`."
- *Condition* = a test as in a where-clause for whether or not the trigger applies.
- *Action* = one or more SQL statements.
- Oracle version and SQL3 version; not in SQL2.
- Differ from checks or SQL2 assertions in that:
    1. Event is programmable, rather than implied by the kind of check.
    2. Condition not available in checks.

Example

Whenever we insert a new tuple into `Sells`, make sure the beer mentioned is also mentioned in `Beers`, and insert it (with a null manufacturer) if not.

```
Sells(bar, beer, price)

CREATE OR REPLACE TRIGGER BeerTrig
AFTER INSERT ON Sells
FOR EACH ROW
WHEN(new.beer NOT IN
            (SELECT name FROM Beers))
    BEGIN
        INSERT INTO Beers(name)
        VALUES(:new.beer);
    END;
.
run
```

Options

1. Can omit `OR REPLACE`. Effect is that it is an error if a trigger of this name exists.

2. `AFTER` can be `BEFORE`.

3. `INSERT` can be `DELETE` or `UPDATE OF` <attribute> `ON`.

4. `FOR EACH ROW` can be omitted, with an important effect: the action is done once for the relation(s) consisting of all changes.

Notes

- More information in on-line document `or-plsql.html`
- There are two special variables `new` and `old`, representing the new and old tuple in the change.
    - ✧ `old` makes no sense in an insert, and `new` makes no sense in a delete.
- Notice: in `WHEN` we use `new` and `old` without a colon, but in actions, a preceding colon is needed.
- The action is a PL/SQL statement.
    - ✧ Simplest form: surround one or more SQL statements with `BEGIN` and `END`.
    - ✧ However, select-from-where has a limited form.
- Dot and `run` cause the definition of the trigger to be stored in the database.
    - ✧ Oracle triggers are elements of the database, like tables or views.

Example

Maintain a list of all the bars that raise their price for some beer by more than $1.

```
Sells(bar, beer, price)

CREATE TRIGGER PriceTrig
AFTER UPDATE OF price ON Sells
FOR EACH ROW
WHEN(new.price > old.price + 1.00)
    BEGIN
        INSERT INTO RipoffBars
        VALUES(:new.bar);
    END;
.
run
```