Still More Operators: Outer Join

- outer join is an extension of the join operation to deal with missing information
  - three types: left outer join, right outer join, and full outer join
  - left outer join computes the natural join, then takes all tuples (rows) in the left relation that did not match on the join attribute and includes them in the result, with all attributes of the right relation padded with null values
  - right outer join is the same, except non-matching tuples in the right relation are included in the result padded with null values
  - full outer join includes all non-matching tuples of both relations appropriately padded
  - see examples in text, p108-109
Still More Operators

• division
  – R/S: given R(A,B) and S(B), then a given tuple \( t \) is in R/S if for all \( s \) in S there exists an \( r \) in R such that \( r.B=s.B \) and \( t.A=r.A \).
  – So tuple \( t \) with attribute \( t.A \) is in the result if and only if R contained tuples \( (t.A, B_1), (t.A, B_2), (t.A, B_3), \ldots \) for every possible value \( B_i \) contained in S.
  – Note that S must be defined on a subset of the attributes of R for the operation to be meaningful.
A Short Interlude: Integrity

• the preceding slides covered chapter three up to section 3.3
• before attacking chapter 4 (SQL), we’re going to make a brief excursion up to chapter 6, touching sections 6.1 - 6.4
• Integrity constraints attempt to enforce data consistency and prevent accidental damage to the database during updates
• We’ve already seen two forms of integrity constraints:
  – key declarations (stipulating that certain attributes form a candidate key for a given entity set)
  – mapping form of a relationship (one-one, one-many, many-many)
Integrity Constraints

• Domain Constraints
  – simplest form of integrity constraint
  – type declarations are one such domain constraint (e.g., integer, floating point, double-precision, fixed length character string).
  – domains can be further restricted (e.g., check clause in SQL can ensure that hourly wages are ? 4.00 dollars)
  – easily tested whenever a new data item is entered into the database
  – extensions like date or currency can be easily supported on a strongly typed programming language
  – Null values can be useful for values to be filled in later, but some attributes may need to be specified as “not Null” (e.g., primary keys cannot have a null value)
Integrity Constraints (2)

- **Key Constraints**
  - keys must have unique values
  - primary key -- a candidate key declared primary
  - unique key -- a candidate key
  - foreign key -- a set of attributes that are a primary key for some other relations
    - foreign keys are an important concept because we need to treat foreign keys differently from other attributes (for example, protecting their uniqueness and insuring *referential integrity*) even though they aren’t a primary key in the current relation
Referential Integrity

- We often want to be able to ensure that an attribute value in a tuple of a relation appears in at least one tuple of another relation. For example:
  
  » EMP(eno, ename, salary)
  » DEPT(dno, dname, floor)
  » WORKS-IN(eno, dno, hours)

- note that eno is a foreign key in WORKS-IN

- We want the following to be true:
  
  • ?_{eno}(WORKS-IN) \& \& ?_{eno}(EMP)  
    (every eno is a real employee)
  • ?_{dno}(WORKS-IN) \& \& ?_{dno}(DEPT)  
    (every dno is a real department)

- SQL allows the declaration of domain/key/referential integrity constraints with the clause **check** in its DDL
Referential Integrity: SQL DDL Example

<table>
<thead>
<tr>
<th>Create table customer</th>
<th>Create table account</th>
<th>Create table depositor</th>
</tr>
</thead>
<tbody>
<tr>
<td>(cust-name char(20) not null,</td>
<td>(account-no char(10) not null,</td>
<td>(cust-name char(20) not null,</td>
</tr>
<tr>
<td>cust-street char(30),</td>
<td>branch-name char(15),</td>
<td>account-no char(10) not null,</td>
</tr>
<tr>
<td>cust-city char(30),</td>
<td>balance number,</td>
<td>primary key char(20) not null,</td>
</tr>
<tr>
<td>primary key (cust-name))</td>
<td>primary key (account-no),</td>
<td>foreign key (cust-name) references</td>
</tr>
<tr>
<td></td>
<td>check (balance ?0))</td>
<td>customer,</td>
</tr>
<tr>
<td>Create table branch</td>
<td></td>
<td>foreign key (account-no) references</td>
</tr>
<tr>
<td>(branch-name char(15) not null,</td>
<td></td>
<td>balance number,</td>
</tr>
<tr>
<td>branch-city char(30),</td>
<td></td>
<td>branch-name char(15),</td>
</tr>
<tr>
<td>assets number,</td>
<td></td>
<td>balance number,</td>
</tr>
<tr>
<td>primary key (branch-name),</td>
<td></td>
<td>balance number,</td>
</tr>
<tr>
<td>check (assets ?0))</td>
<td></td>
<td>branch-name char(15),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreign key (account-no) references</td>
</tr>
<tr>
<td></td>
<td></td>
<td>account-no char(10) not null,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>primary key (cust-name),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreign key (account-no) references</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreign key (account-no) references</td>
</tr>
<tr>
<td></td>
<td></td>
<td>account-no char(10) not null,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>primary key (cust-name),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreign key (account-no) references</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreign key (account-no) references</td>
</tr>
</tbody>
</table>
Referential Integrity and Database Modifications

- Database modifications may violate referential integrity
- Insertion: inserting a value into the referencing relation that is not in the referenced relation
- Deletion: deleting the last example of a given value in the referenced relation and leaving that value in the referencing one
  - proper handling may lead to cascading deletions
- Update to the referencing relation (constraints as Insertion)
- Update to the referenced relation (constraints as Deletion)
Assertions

- An **assertion** is an arbitrary expression that the database must always satisfy
  - e.g., student GPA > 2.8, or sum(all-charges) < credit-line
  - Domain constraints and referential integrity constraints are special forms of assertion that are easy to test
  - SQL supports assertions as follows:
    ```sql
    create assertion <assertion-name> check <predicate>
    ```
  - When an assertion is made the system checks it for validity. If it is validated, every future modification of the database is checked against the assertion and allowed only if it is not violated.
  - This can be very expensive if assertions are complex or numerous
Triggers

- A trigger is a statement that the system executes automatically as a side effect of an update to the database.
- A trigger has two parts:
  - condition under which it is executed
  - actions to be taken if it is executed
- Example: instead of having an assertion “balance ≤ 0” for a checking account, use a trigger on negative balances that sets the balance to zero and creates a new loan for the amount of the overdraft
- Triggers make the system reactive
- Triggers are also called active rules
- Like Assertions, Triggers can be very expensive.
Trigger Example

```sql
define trigger overdraft on update of account T
  (if new T.balance < 0 then (insert into borrow values
    (T.branch.name, T.account-number,
     T-customer-name, - new T.balance)
    update deposit S
    set S.balance = 0
    where S.account-number = T.account-number))
```

(note: SQL syntax given here is slightly different from that in the text, p235)
SQL (Structured Query Language)  
(Astrahan, Gray, Lindsay, Selinger, …)

- Most common and influential commercial query language; well established as the industry standard query language for relational databases
- Developed (as “Sequel”) at the IBM Research Lab in San Jose in the early 70s
- Four basic commands
  - select
  - insert
  - delete
  - update
- Result of each query is a relation
SQL Example

```
select e.name
from emp e
where e.age > 30;
```

- e is a tuple variable ranging over the emp relation
- a tuple variable followed by a “.” and an attribute is an *indexed tuple variable* and specifies the corresponding attribute of the tuple, very similarly to in many programming languages
- what follows the keyword `select` is the target list
- what follows `from` is the *tuple variable list* and consists of a list of relations and variable names
- what follows `where` is the *qualification clause*; an arbitrary boolean expression
SQL

- Basic format of the `select` command
  ```sql
  select [distinct] target_list
  from tuple_variable_list
  where qualification
  [order by target_list_subset];
  ```

- Semantics
  - evaluate qualification: select the subset of the cartesian product of the ranges of the tuple variables that satisfy the qualification
  - evaluate target list: eliminate columns that are not in the target list
  - prepare the result as a relation with columns according to the target list
  - if `distinct` is used, eliminate duplicate tuples
  - if `order by` is used, sort the result accordingly
SQL: some example queries

• We will give a number of simple query examples using the following relational schema:
  
sailors(sid, sname, rating)
boats(bid, bname, colour)
reserve(sid, bid, date)

(1) Find the names of sailors who have reserved boat #2
  
  select s.sname
from sailors s, reserve r
where s.sid=r.sid and r.bid=2
SQL: example queries (2)

(2) Find the names of sailors who have reserved a red boat
    select s.sname
    from sailors s, reserve r, boats b
    where s.sid=r.sid and r.bid=b.bid and b.colour="red"

(3) Find the colours of all boats reserved by Pat
    select b.colour
    from sailors s, reserve r, boats b
    where s.sname="Pat" and s.sid=r.sid and r.bid=b.bid
(4) Find the names of sailors who have reserved at least one boat

```sql
select s.sname
from sailors s, reserve r
where s.sid=r.sid
```

(5) Find the names of sailors who have reserved a red or a green boat

```sql
select s.sname
from sailors s, reserve r, boats b
where s.sid=r.sid and r.bid=b.bid and
(b.colour="red" or b.colour="green")
```
(6) Find the names of sailors who have reserved a red and a green boat

```sql
select s.sname
from sailors s, reserve r, boats b, reserve r2, boats b2
where s.sid=r.sid and r.bid=b.bid and b.colour="red"
and s.sid=r2.sid and r2.bid=b2.bid and
b2.colour="green")
```

Note: in the above query if sailor Pat has reserved one green boat and two red ones, the name Pat will appear twice in the results. To avoid that, use the keyword `distinct` in the select line, as in:

```sql
select distinct s.sname
```