Announcements

• Test 1 will cover material from:
  • lecture material from weeks 1 through 4
  • activities from weeks 1 through 4
  • demo sessions: 1, 2, 3, and 4
  • assignments: 1, 2, 3, and 4

• Demos 1 and 2 are graded and entered into d2l
• Demo 3 will soon be entered

• Grading scheme for demos – written & live questions:
  3 points – meets expectations; answer is correct
  2 points – needs improvement; answer is on the right track
  1 point – unsatisfactory; answer is not close to right
  0 points – question was not attempted

• Assignment 1 is graded and will be returned tonight
DATABASE MODIFICATIONS AND CONSTRAINTS IN SQL

Lois Delcambre
Winter 2013
row value constructors in SQL:1999

- A row value constructor allows you to create a row “on the fly” for example:

```sql
WHERE (E.Lname, E.Fname) = (S.Lname, S.Fname)
```

Each of these are rows…

and the equality comparison is doing a pair-wise comparison on the two rows.
table value constructors in SQL:1999

• You can also create a table “on the fly”

VALUES row-value-expr, ..., row-value-expr

Example:
INSERT INTO movie_stars
VALUES ("Rocky Horror Picture Show", 1977, "Curry, Tim")

• The part circled in GREEN is a row-value expression.

• The part shown in RED font is a table value constructor
What about row/table constructors in relational algebra?

If you need to have either a row or a relation, you can just define it and then use it in relational algebra.

You could say something like:
Let \( R = \{ ("John", 5, "male"), ("Sue", 6, "female") \} \)

or let \( R \) be the table

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>5</td>
<td>male</td>
</tr>
<tr>
<td>Sue</td>
<td>6</td>
<td>female</td>
</tr>
</tbody>
</table>

and then use \( R \) in expressions … like \( R \times \text{Student} \) …or whatever
Row value constructors in SQL

- in the from clause:

```
select *
from sailors, (values (1), (2)) as x
```

This is a table-value constructor. This creates a temporary table (with no name) that has just one attribute. The table has two rows: the one with value 1 and the one with value 2.

What answer would you expect from this query?
Inserting rows into a table

```sql
insert into <relation-name> (a₁, a₂, a₃, ..., aₙ)
values (v₁, v₂, v₃, ..., vₙ);
```

If an attribute is not listed, the default value will be used.
You can list more than one row-value expression (like prior slide).

```sql
insert into <relation-name> values (v₁, v₂, v₃, ..., vₙ);
```
No attributes listed; values must be in the proper order (as defined).

```sql
insert into sailors(sid)
    Select DISTINCT sid from reserves
    Where sid not in
        (Select sid from sailors);
```
You can use a subquery to deliver rows to be inserted.
Deleting rows from a table

delete from <tablename> where <condition>;

The <condition> specifies what tuple(s) to be deleted.

To delete several tuples, use a condition satisfied by several tuples

delete from agent where salary > 8000;
Updating rows in a table

update <tablename>  set <attribute> = value
  where <condition>;

Here’s an example:

update agent
set address = ‘111 E Portlandia Ave’
where agent_id = 99
Constraints in a DB

• If we don’t have any constraints, then the table is kind of like a spreadsheet – we can put any data in any table. (We only have to use the right data type – to match the attribute definition in the create table statement.)

• So, what mechanisms do we have for constraints in SQL:
  • primary key and unique constraints for a table
  • check constraints on an attribute value that can appear (beyond just type constraints) including NOT NULL constraint
  • check constraints on a tuple
  • assertions (not yet implemented in postgresql)
  • triggers (that can include code) similar to DB procedures
Primary key and Unique Constraints

• We’ve already seen these; one or several attributes can be declared to be a primary key.

• One or several attributes can be declared to be unique.

• Look at the key for the languagerel in the Spy database to see a key that consists of two attributes: (lang_id, agent_id)

• If a table has two keys, then you can declare one to be a primary key and the other to be unique. You can also declare two different unique constraints.
Tables can have keys and foreign keys

A key for a table: one or more attributes whose values uniquely identify the rows in the table (for all future data). sid is unique for all sailors. The combination of (sailor, boat) is unique for all reserves.

A foreign key in a table: one or more attributes whose values must match the values of a key in some table. reserves.sailor is a foreign key that references sailors.sid.
Foreign key can’t be violated (referential integrity)

• For a table that references another table (like bid in reserves), it must point to a valid row (e.g., to a bid that is in the boats table).

• The second row has an invalid bid (107) because there is not row in the boats table with 107.

(Only part of the reserves table is shown here.)
Foreign keys in a schema

create table sailors (  
  sid integer primary key,  
sname character varying (30),  
rating integer,  
age integer)
create table reserves (  
  sid integer references sailors (sid),  
bid integer,  
day date,  
primary key (sid, bid, day))
Foreign key constraints

create table reserves (sid integer references sailors (sid), bid integer, day date);
create table reserves (sid integer bid integer, day date)
foreign key (sid) references sailors (sid);
create table reserves (sid integer bid integer, day date)
constraint fk1 foreign key (sid) references sailors (sid);
When does a foreign key need to be checked?

cREATE TABLE sailors (  
  sid integer primary key,  
sname character varying (30));

CREATE TABLE reserves (  
  sid integer references sailors (sid),  
bid integer,  
day date);

When you insert a sailor?  When you insert a reservation?  
When you delete a sailor?  When you delete a reservation?  
When you modify an sid in sailor?  When you modify an sid in a reservation?
Naming constraints

create table sailors (sid integer primary key, sname);
create table sailors (sid integer constraint pk1 primary key, sname);
create table reserves (sid integer constraint fk2 references sailors(sid), bid integer, day date);
Three policies for FK enforcement for deletes and updates

create table reserves(
    sid integer references sailors(sid)
    on delete cascade
    on update cascade,
    bid integer);
create table reserves(
    sid integer references sailors(sid)
    on delete set null
    on update set null,
    bid integer);

Note: the default policy is to reject the update if FK is violated.
Named constraints can be dropped or added to tables

```
alter table <tablename> drop constraint <constraintname>

alter table <tablename> add constraint <constraintname>
    primary key (a1, a2, a3, ..)

alter table <tablename> add constraint <constraintname>
    foreign key (a1, a2, …) references <tablename>(b1, b2, …)
```
Constraints on attribute values

• not null
• primary key
• references

• check (condition)
  • check (salary > 20000)
  • check (gender in ('f', 'm'))
  • check (agent_id in <subquery>)
    This is NOT allowed in postgresql; you can’t use a subquery.

• When are these constraints checked?
Constraints on tuples

At the end of a create table statement, you can put:

check (condition)

This condition is like any WHERE clause; you can mention any of the attributes in this table.

SQL allows a subquery in the condition but postgresql does not. Read the book to see the risk; postgresql has made a reasonable choice.
Tuple-based check constraint

create table sailors(
    sid integer constraint pksid primary key,  
    sname character varying (30) not null,    
    rating integer,                         
    age integer                             
check (rating < age));
create table sailors(
    sid integer constraint pksid primary key, 
    sname character varying (30) not null,    
    rating integer,                          
    age integer                             
constraint ratingsmall check (rating < age));
Assertions in SQL

CREATE ASSERTION <assertionname>
CHECK (condition);

Condition can be any condition; can involve arbitrary subqueries. Can use EXISTS, NOT EXISTS, etc.

Write a condition for an assertion that will check for violation of a foreign key.

When is an assertion checked? How often would that constraint be checked? Compare that to how often (and for how much data) a foreign key constraint would be checked.
Triggers in SQL

Trigger = ECA rules = event/condition/action rules

You can trigger – before or after some triggering event.

A triggering event is: an insert, delete, or update for a table. (We’ll talk about transactions later.)

When awakened, a trigger tests a condition. It it’s true, then the trigger action is executed.

In postgresql, the action must be a user-defined function.
STORED FUNCTIONS & MODULES IN POSTGRES

by Neena Maldikar
PSM (Persistent Stored Modules)

- An example of a stored function in PostgreSQL:

```sql
CREATE OR REPLACE FUNCTION delete_movies(
    IN in_title CHAR(50),
    IN in_year INTEGER
) 
RETURNS void AS $$
BEGIN
    DELETE FROM movies
    WHERE title = in_title
    AND year = in_year;
END;
$$ LANGUAGE plpgsql;
```

- To call the Stored Function, we can query as follows:

```sql
SELECT delete_movies('Hugo', 2011)
```
PSM (Persistent Stored Modules)

- An example of a function in Postgres that returns a value

```sql
CREATE OR REPLACE FUNCTION get_genre(title_in char, year_in int) RETURNS varchar AS $$
DECLARE
    genre_ret varchar;
BEGIN
    SELECT genre INTO genre_ret
    FROM movies
    WHERE title = title_in
    AND year = year_in;
    RETURN genre_ret;
END;
$$ LANGUAGE plpgsql;
```

- To call the Stored Function, we can query as follows:

```sql
SELECT get_genre('Star Wars', 1977)
```
PSM (Persistent Stored Modules)

- An example of a function in Postgres that returns multiple results. (It must be declared to return SETOF some type.)

```sql
CREATE OR REPLACE FUNCTION get_movies()
RETURNS SETOF movies AS $$
DECLARE
    row RECORD;
BEGIN
    FOR row IN (Select * From movies ) LOOP
        RETURN NEXT row;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

- To call the Stored Function, we can query as follows:

```sql
SELECT * FROM get_movies()
```
PSM (Persistent Stored Modules)

- An example of a function in Postgres that uses IF – THEN – END IF and inserts rows.

```sql
CREATE OR REPLACE FUNCTION update_movies( )
RETURNS integer AS $$
DECLARE
    row RECORD;
    counter INTEGER;
BEGIN
    counter = 0;
    FOR row IN SELECT * FROM movies
    LOOP
        IF (row.length > 100) THEN
            INSERT INTO movies1(title, year, genre)
            VALUES (row.title, row.year, row.genre);
            counter = counter + 1;
        END IF;
    END LOOP;
    RETURN counter;
END;
$$ LANGUAGE plpgsql;
```
PSM (Persistent Stored Modules)

• An example of a function that returns cursors: (We have to use refcursor return type.)

```sql
CREATE OR REPLACE FUNCTION get_moviescur(
    ref ref cursor
) RETURNS refcursor AS $$
BEGIN
    OPEN ref FOR SELECT * FROM movies;
    RETURN ref;
END;
$$ LANGUAGE plpgsql;
```

• To call the Stored Function, we can query as follows:

```sql
SELECT show_movies('movies_cur');
FETCH ALL IN "movies_cur";
```
Triggers

Classes (class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)

When a new class is inserted into Classes, also insert a ship with the name of that class and a NULL launch date.

CREATE TRIGGER AddShipTrigger
AFTER INSERT ON Classes
REFERENCING
    NEW ROW AS NewRow
FOR EACH ROW
INSERT INTO Ships(name, class, launched)
    VALUES (NewRow.class, NewRow.class, NULL);
CREATE FUNCTION shipTrigger()
RETURNS trigger AS $$
BEGIN
    INSERT INTO Ships(name, class , launched)
    VALUES (New.class, New.class, NULL);
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER AddShipTrigger
AFTER INSERT ON Classes
FOR EACH ROW EXECUTE PROCEDURE shipTrigger();

Triggers – Implementation in Postgres
CREATE TABLE Classes (  
    class VARCHAR(50) PRIMARY KEY,
    type CHAR (2),
    country VARCHAR(50),
    numGuns INT,
    bore INT,
    displacement INT)

CREATE TABLE Ships (  
    name VARCHAR(50) PRIMARY KEY,
    class VARCHAR(50),
    launched INT,
    FOREIGN KEY (class) REFERENCES Classes (class)
);  

INSERT INTO classes(class , type , country, numGuns, bore, displacement)VALUES('Iowa','bb','usa', 9, 16, 46000);
References:

- [http://www.eioba.com/a/1ign/a-basic-introduction-to-postgres-stored-procedures](http://www.eioba.com/a/1ign/a-basic-introduction-to-postgres-stored-procedures)


- [http://www.postgresql.org/docs/8.0/static/plpgsql.html](http://www.postgresql.org/docs/8.0/static/plpgsql.html)