SQL: Updating the Data

- Data Modification in SQL
- Insertion
- Bulk Insertion of Data
- Deletion
- Semantics of Deletion
- Updates

❖ Data Modification in SQL

We have seen statements to modify table meta-data (in DB catalog):

- **`CREATE TABLE`** ... add new, initially empty, table to DB
- **`DROP TABLE`** ... remove table data (all tuples) and meta-data
- **`ALTER TABLE`** ... change meta-data of table (e.g add constraints)

SQL also provides statements for modifying data in tables:

- **`INSERT`** ... add a new tuple(s) into a table
- **`DELETE`** ... remove tuples from a table (via condition)
- **`UPDATE`** ... modify values in existing tuples (via condition)

Constraint checking is applied automatically on any change.

Operation fails (no change to DB) if any constraint check fails

❖ Insertion

Add new tuples via the **INSERT** operation:

```
INSERT INTO RelationName
VALUES (val1, val2, val3, ...)

INSERT INTO RelationName(Attr1, Attr2, ...)
VALUES (valForAttr1, valForAttr2, ...)

INSERT INTO RelationName
VALUES Tuple1, Tuple2, Tuple3, ...
```

The first two add a single new tuple into *RelationName*.

The last form adds multiple tuples into *RelationName*.

❖ Insertion (cont)

`INSERT INTO` *R* **`VALUES (`**v_1`,`v_2`,...)`**

- values must be supplied for all attributes of *R*
- in same order as appear in **`CREATE TABLE`** statement
- special value **`DEFAULT`** forces default value or **`NULL`**

`INSERT INTO` *R*`(`A_1`,`A_2`,...)` **`VALUES (`**v_1`,`v_2`,...)`

- can specify any subset of attributes of *R*
- values must match attribute specification order
- unspecified attributes are assigned default or null

❖ Insertion (cont)

Example: Add the fact that Justin likes 'Old'.

```
INSERT INTO Likes VALUES ('Justin','Old');
-- or --
INSERT INTO Likes(drinker,beer) VALUES('Justin','Old');
-- or --
INSERT INTO Likes(beer,drinker) VALUES('Old','Justin');
```

Example: Add a new beer with unknown style.

```
INSERT INTO Beers(name,brewer)
       VALUES('Mysterio','Hop Nation');
-- which inserts the tuple ...
('Mysterio', 'Hop Nation', null)
```

❖ Insertion (cont)

Example: insertion with default values

```
ALTER TABLE Likes
   ALTER COLUMN beer SET DEFAULT 'New';
ALTER TABLE Likes
   ALTER COLUMN drinker SET DEFAULT 'Joe';

INSERT INTO Likes(drinker) VALUES('Fred');
INSERT INTO Likes(beer) VALUES('Sparkling Ale');

-- inserts the two new tuples ...
('Fred', 'New')
('Joe', 'Sparkling Ale')
```

❖ Insertion (cont)

Example: insertion with insufficient values.

E.g. specify that drinkers' phone numbers cannot be **`NULL`**.

```
ALTER TABLE Drinkers
   ALTER COLUMN phone SET NOT NULL;
```

Then try to insert a drinker whose phone number we don't know:

```
INSERT INTO Drinkers(name,addr) VALUES ('Zoe','Manly');

ERROR:  null value in column "phone" violates
                              not-null constraint
DETAIL:  Failing row contains (Zoe, Manly, null).
```

❖ Bulk Insertion of Data

Tuples may be inserted individually:

```
insert into Stuff(x,y,s) values (2,4,'green');
insert into Stuff(x,y,s) values (4,8,null);
insert into Stuff(x,y,s) values (8,null,'red');
...
```

but this is tedious if 1000's of tuples are involved.

It is also inefficient

- all relevant constraints are checked on insertion of each tuple

So, most DBMSs provide non-SQL methods for bulk insertion

❖ Bulk Insertion of Data (cont)

Bulk insertion methods typically ...

- use a compact representation for each tuple
- "load" all tuples without constraint checking
- do all constraint checks at the end
- if any tuples fail checks, none are inserted

Example: PostgreSQL's **copy** statement:


```
COPY Stuff(x,y,s) FROM stdin;
2       4       green
4       8       \N
8       \N      red
\.
```

Can also copy from a named file (but must be readable by PostrgeSQL server)

❖ Deletion

Removing tuples is accomplished via **`DELETE`** statement:

```
DELETE FROM Relation
WHERE  Condition
```

Removes all tuples from *Relation* that satisfy *Condition*.

Example: Justin no longer likes Sparkling Ale.

```
DELETE FROM Likes
WHERE drinker = 'Justin'
      AND beer = 'Sparkling Ale';
```

Special case: Make relation *R* empty.

```
DELETE FROM R;     or     DELETE FROM R WHERE true;
```

❖ Deletion (cont)

Example: remove all expensive beers from sale.

```
DELETE FROM Sells WHERE price >= 5.00;
```

Example: remove all beers with unknown style

```
DELETE FROM Beers WHERE style IS NULL;
```

This fails* if such Beers are referenced from other tables

E.g. such Beers are liked by someone or sold in some bar

* no beers are removed, even if some are not referenced

❖ Semantics of Deletion

Method A for **`DELETE FROM`** *R* **`WHERE`** *Cond*:

```
FOR EACH tuple T in R DO
    IF T satisfies Cond THEN
        remove T from relation R
    END
END
```

Method B for **`DELETE FROM`** *R* **`WHERE`** *Cond*:

```
FOR EACH tuple T in R DO
    IF T satisfies Cond THEN
        make a note of this T
    END
END
FOR EACH noted tuple T DO
    remove T from relation R
END
```

❖ Semantics of Deletion (cont)

Does it matter which method the DBMS uses?

For most cases, the same tuples would be deleted

But if *Cond* involes a query on the table *R*

- the result of *Cond* might change as the deletion progresses
- so Method A might delete less tuples than Method B

E.g.

```
DELETE FROM Beers
WHERE (SELECT count(*) FROM Beers) > 10;
```

Method A deletes beers until there are only 10 left
Method B deletes all beers if there were more than 10 to start with

❖ Updates

The **`UPDATE`** statement allows you to ...

- modify values of specified attributes in specified tuples of a relation

```
UPDATE R
SET    List of assignments
WHERE  Condition
```

Each tuple in relation *R* that satisfies *Condition* is affected

Assignments may:

- assign constant values to attributes,
  e.g. **`SET price = 2.00`**
- use existing values in the tuple to compute new values,
  e.g. **`SET price = price * 0.5`**

❖ Updates (cont)

Example: Adam changes his phone number.

```
UPDATE Drinkers
SET    phone = '9385-2222'
WHERE  name = 'Adam';
```

Example: John moves to Coogee.

```
UPDATE Drinkers
SET    addr = 'Coogee',
       phone = '9665-4321'
WHERE  name = 'John';
```

❖ Updates (cont)

Examples that modify many tuples ...

Example: Make $6 the maximum price for beer.

```
UPDATE Sells
SET    price = 6.00
WHERE  price > 6.00;
```

Example: Increase beer prices by 10%.

```
UPDATE Sells
SET    price = price * 1.10;
```

Updates all tuples (as if **`WHERE true`**)

Produced: 28 Sep 2020