SQL

Lecture #5

The Main Thing to Remember

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

- Later: the devil is in the details

* In SELECT clauses

- When there is one relation in the FROM clause, * in the SELECT clause stands for "all attributes of this relation."
- Example using Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

Renaming Attributes

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute.
- Example based on Beers(name, manf):

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

Result of Query:

beer	manf
‘Bud’	‘Anheuser-Busch’
‘Bud Lite’	‘Anheuser-Busch’
‘Michelob’	‘Anheuser-Busch’

Expressions in SELECT Clauses

- Any expression that makes sense can appear as an element of a SELECT clause.
- Example: from Sells(bar, beer, price):

```
SELECT bar, beer,
       price * 120 AS priceInYen
FROM Sells;
```

Result of Query

bar	beer	priceInYen
Joe's	Bud	300
Sue's	Miller	360
…	…	…

Null Values

NULL Values

- Tuples in SQL relations can have NULL as a value for one or more components.
- Meaning depends on context. Two common cases:
  - *Missing value* : e.g., we know Joe's Bar has some address, but we don't know what it is.
  - *Inapplicable* : e.g., the value of attribute *spouse* for an unmarried person.

How to Handle NULL?

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
c	NULL

Sells(bar, beer, price)

t,	x,	5
t,	y,	4
u,	x,	7
u,	y,	8

- Find all drinkers who frequent bar t
- Find all drinkers who frequent bars that sell beer x.
- So should we be conservative or liberal?

RDBMSs Today Are Conservative

Comparing NULL's to Values

- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.
- When any value is compared with NULL, the truth value is UNKNOWN.
- But a query only produces a tuple in the answer if its truth value for the WHERE clause is TRUE (not FALSE or UNKNOWN).

This Can Lead to Surprising Scenarios

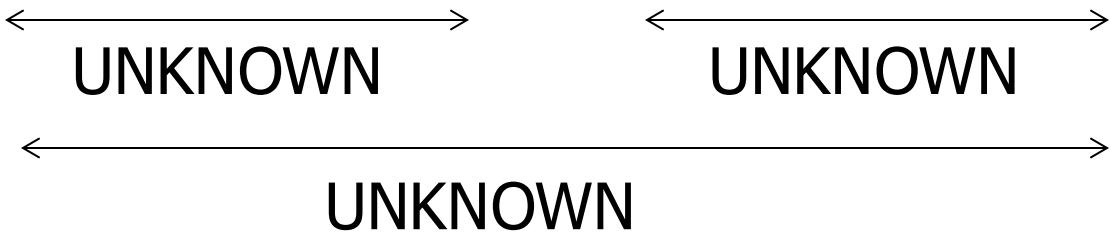
- From the following Sells relation:

bar	beer	price
Joe's Bar	Bud	NULL

SELECT bar

FROM Sells

WHERE price < 2.00 OR price >= 2.00;



Another Example

Unexpected behavior:

```
SELECT *
FROM   Person
WHERE  age < 25  OR  age >= 25
```

Some Persons are not included !

Testing for Null

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *
FROM   Person
WHERE  age < 25  OR  age >= 25 OR age IS NULL
```

Now it includes all Persons

Subqueries

Subqueries

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places, including FROM and WHERE clauses.
- Example: in place of a relation in the FROM clause, we can place another query, and then query its result.
  - Better use a tuple-variable to name tuples of the result.

Subqueries That Return One Tuple

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.
  - Usually, the tuple has one component.
  - Also typically, a single tuple is guaranteed by keyness of attributes.
  - A run-time error occurs if there is no tuple or more than one tuple.

Example

- From Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.
- Two queries would surely work:
  1. Find the price Joe charges for Bud.
  2. Find the bars that serve Miller at that price.

Query + Subquery Solution

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
    price = (SELECT price
             FROM Sells
             WHERE bar = 'Joe''s Bar'
                 AND beer = 'Bud');
```

The price at which Joe sells Bud

Boolean Operators IN, EXISTS, ANY, ALL

The IN Operator

- <tuple> IN <relation> is true if and only if the tuple is a member of the relation.
  - <tuple> NOT IN <relation> means the opposite.
- IN-expressions can appear in WHERE clauses.
- The <relation> is often a subquery.

Example

- From Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Fred likes.

```
SELECT *
FROM Beers
WHERE name IN (SELECT beer
               FROM Likes
               WHERE drinker = 'Fred');
```

The set of beers Fred likes

The Exists Operator

- EXISTS(<relation>) is true if and only if the <relation> is not empty.
- Being a boolean-valued operator, EXISTS can appear in WHERE clauses.
- Example: From Beers(name, manf), find those beers that are the unique beer by their manufacturer.

Example Query with EXISTS

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS (
    SELECT *
    FROM Beers
    WHERE manf = b1.manf AND
          name <> b1.name);
```

Notice scope rule: manf refers to closest nested FROM with a relation having that attribute.

Set of beers with the same manf as b1, but not the same beer

Notice the SQL "not equals" operator

The Operator ANY

- *x* = ANY(<relation>) is a boolean condition meaning that *x* equals at least one tuple in the relation.
- Similarly, = can be replaced by any of the comparison operators.
- Example: *x* >= ANY(<relation>) means *x* is not smaller than all tuples in the relation.
  - Note tuples must have one component only.

The Operator ALL

- Similarly, *x* <> ALL(<relation>) is true if and only if for every tuple *t* in the relation, *x* is not equal to *t*.
  - That is, *x* is not a member of the relation.
- The <> can be replaced by any comparison operator.
- Example: *x* >= ALL(<relation>) means there is no tuple larger than *x* in the relation.

Example

- From Sells(bar, beer, price), find the beer(s) sold for the highest price.

```
SELECT beer
FROM Sells
WHERE price >= ALL(
    SELECT price
    FROM Sells);
```

price from the outer Sells must not be less than any price.

Defining a Database Schema

Defining a Database Schema

- A database schema comprises declarations for the relations (“tables”) of the database.
- Many other kinds of elements may also appear in the database schema, including views, indexes, and triggers, which we’ll mention later.

Declaring a Relation

- Simplest form is:

  CREATE TABLE <name> (

  <list of elements>

  );

- And you may remove a relation from the database schema by:

  DROP TABLE <name>;

Elements of Table Declarations

- The principal element is a pair consisting of an attribute and a type.
- The most common types are:
  - INT or INTEGER (synonyms).
  - REAL or FLOAT (synonyms).
  - CHAR(*n*) = fixed-length string of *n* characters.
  - VARCHAR(*n*) = variable-length string of up to *n* characters.

Example: Create Table

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer        VARCHAR(20),
    price       REAL
);
```

Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE.
- There are a few distinctions to be mentioned later.

Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

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

Declaring Multiattribute Keys

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement.
- This form is essential if the key consists of more than one attribute.
  - May be used even for one-attribute keys.

Example: Multiattribute Key

- The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer    VARCHAR(20),
    price   REAL,
    PRIMARY KEY (bar, beer)
);
```

PRIMARY KEY Versus UNIQUE

- The SQL standard allows DBMS implementers to make their own distinctions between PRIMARY KEY and UNIQUE.
  - Example: some DBMS might automatically create an *index* (data structure to speed search) in response to PRIMARY KEY, but not UNIQUE.

Required Distinctions

- However, standard SQL requires these distinctions:
  1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
  2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

Other Declarations for Attributes

- Two other declarations we can make for an attribute are:
  1. NOT NULL means that the value for this attribute may never be NULL.
  2. DEFAULT <value> says that if there is no specific value known for this attribute's component in some tuple, use the stated <value>.

Example: Default Values

```
CREATE TABLE Drinkers (
   name CHAR(30) PRIMARY KEY,
   addr CHAR(50)
        DEFAULT '123 Sesame St.',
   phone CHAR(16)
);
```

Effect of Defaults -- 1

- Suppose we insert the fact that Sally is a drinker, but we know neither her address nor her phone.
- An INSERT with a partial list of attributes makes the insertion possible:

```
INSERT INTO Drinkers(name)
VALUES('Sally');
```

Effect of Defaults -- 2

- But what tuple appears in Drinkers?

name	addr	phone
‘Sally’	‘123 Sesame St’	NULL

- If we had declared phone NOT NULL, this insertion would have been rejected.

Adding Attributes

- We may change a relation schema by adding a new attribute ("column") by:

  ALTER TABLE <name> ADD

  <attribute declaration>;

- Example:

```
ALTER TABLE Bars ADD
phone CHAR(16)DEFAULT 'unlisted';
```

Deleting Attributes

- Remove an attribute from a relation schema by:

  ALTER TABLE <name>

  DROP <attribute>;

- Example: we don't really need the license attribute for bars:

```
ALTER TABLE Bars DROP license;
```

Database Modification

Database Modifications

- A modification command does not return a result as a query does, but it changes the database in some way.
- There are three kinds of modifications:
  1. *Insert* a tuple or tuples.
  2. *Delete* a tuple or tuples.
  3. *Update* the value(s) of an existing tuple or tuples.

Insertion

- To insert a single tuple:

  INSERT INTO <relation>

  VALUES (<list of values>);

- Example: add to Likes(drinker, beer) the fact that Sally likes Bud.

```
INSERT INTO Likes
VALUES('Sally', 'Bud');
```

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes.
- There are two reasons to do so:
  1. We forget the standard order of attributes for the relation.
  2. We don't have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

Example: Specifying Attributes

- Another way to add the fact that Sally likes Bud to Likes(drinker, beer):

```
INSERT INTO Likes(beer, drinker)
VALUES('Bud', 'Sally');
```

Inserting Many Tuples

- We may insert the entire result of a query into a relation, using the form:

  INSERT INTO <relation>

  (<subquery>);

Example: Insert a Subquery

- Using Frequents(drinker, bar), enter into the new relation PotBuddies(name) all of Sally's "potential buddies," i.e., those drinkers who frequent at least one bar that Sally also frequents.

Solution

The other drinker

Pairs of Drinker tuples where the first is for Sally, the second is for someone else, and the bars are the same.

```
INSERT INTO PotBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = 'Sally' AND
   d2.drinker <> 'Sally' AND
   d1.bar = d2.bar
);
```

Deletion

- To delete tuples satisfying a condition from some relation:

  DELETE FROM <relation>

  WHERE <condition>;

Example: Deletion

- Delete from Likes(drinker, beer) the fact that Sally likes Bud:

```
DELETE FROM Likes
WHERE drinker = 'Sally' AND
    beer = 'Bud';
```

Example: Delete all Tuples

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.

Example: Delete Many Tuples

- Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

```
DELETE FROM Beers b
WHERE EXISTS (
    SELECT name FROM Beers
    WHERE manf = b.manf AND
        name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

Semantics of Deletion -- 1

- Suppose Anheuser-Busch makes only Bud and Bud Lite.
- Suppose we come to the tuple *b* for Bud first.
- The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
- Now, When *b* is the tuple for Bud Lite, do we delete that tuple too?

Semantics of Deletion -- 2

- The answer is that we *do* delete Bud Lite as well.
- The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied in the original relation.
  2. Delete the marked tuples.

Updates

- To change certain attributes in certain tuples of a relation:

  UPDATE <relation>

  SET <list of attribute assignments>

  WHERE <condition on tuples>;

Example: Update

- Change drinker Fred's phone number to 555-1212:

```
UPDATE Drinkers
SET phone = '555-1212'
WHERE name = 'Fred';
```

Example: Update Several Tuples

- Make $4 the maximum price for beer:

```
UPDATE Sells
SET price = 4.00
WHERE price > 4.00;
```

Constraints

Foreign Keys

Local and Global Constraints

Constraints and Triggers

- A *constraint* is a relationship among data elements that the DBMS is required to enforce.
  - Example: key constraints.
- *Triggers* are only executed when a specified condition occurs, e.g., insertion of a tuple.
  - Easier to implement than many constraints.

Kinds of Constraints

- Keys.
- Foreign-key, or referential-integrity.
- Value-based constraints.
  - Constrain values of a particular attribute.
- Tuple-based constraints.
  - Relationship among components.
- Assertions: any SQL boolean expression.

Foreign Keys

- Consider Relation Sells(bar, beer, price).
- We might expect that a beer value is a real beer --- something appearing in Beers.name .
- A constraint that requires a beer in Sells to be a beer in Beers is called a *foreign -key* constraint.

Expressing Foreign Keys

- Use the keyword REFERENCES, either:
  1. Within the declaration of an attribute, when only one attribute is involved.
  2. As an element of the schema, as:

  FOREIGN KEY (<list of attributes>)

  REFERENCES <relation> (<attributes>)
- Referenced attributes must be declared PRIMARY KEY or UNIQUE.

Example: With Attribute

```
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 );
```

Example: As Element

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

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

Enforcing Foreign-Key Constraints

- If there is a foreign-key constraint from attributes of relation *R* to the primary key of relation *S*, two violations are possible:
  1. An insert or update to *R* introduces values not found in *S*.
  2. A deletion or update to S causes some tuples of *R* to "dangle."

Actions Taken -- 1

- Suppose *R* = Sells, *S* = Beers.
- An insert or update to Sells that introduces a nonexistent beer must be rejected.
- A deletion or update to Beers that removes a beer value found in some tuples of Sells can be handled in three ways.

Actions Taken -- 2

- The three possible ways to handle beers that suddenly cease to exist are:
  1. *Default* : Reject the modification.
  2. *Cascade* : Make the same changes in Sells.
     - Deleted beer: delete Sells tuple.
     - Updated beer: change value in Sells.
  3. *Set NULL* : Change the beer to NULL.

Example: Cascade

- Suppose we delete the Bud tuple from Beers.
  - Then delete all tuples from Sells that have beer = ’Bud’.
- Suppose we update the Bud tuple by changing ’Bud’ to ’Budweiser’.
  - Then change all Sells tuples with beer = ’Bud’ so that beer = ’Budweiser’.

Example: Set NULL

- Suppose we delete the Bud tuple from Beers.
  - Change all tuples of Sells that have beer = ’Bud’ to have beer = NULL.
- Suppose we update the Bud tuple by changing ’Bud’ to ’Budweiser’.
  - Same change.

Choosing a Policy

- When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates.
- Follow the foreign-key declaration by:

ON [UPDATE, DELETE][SET NULL CASCADE]

- Two such clauses may be used.
- Otherwise, the default (reject) is used.

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 );
```

Attribute-Based Checks

- Put a constraint on the value of a particular attribute.
- CHECK(<condition>) must be added to the declaration for the attribute.
- The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery.

Example

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

Timing of Checks

- An attribute-based check is checked only when a value for that attribute is inserted or updated.
  - Example: CHECK (price <= 5.00) checks every new price and rejects it if it is more than $5.
  - Example: CHECK (beer IN (SELECT name FROM Beers)) not checked if a beer is deleted from Beers (unlike foreign-keys).

Tuple-Based Checks

- CHECK (<condition>) may be added as another element of a schema definition.
- The condition may refer to any attribute of the relation, but any other attributes or relations require a subquery.
- Checked on insert or update only.

Example: Tuple-Based Check

- 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)
);
```

Assertions

- These are database-schema elements, like relations or views.
- Defined by:

    CREATE ASSERTION <name>

        CHECK (<condition>);

- Condition may refer to any relation or attribute in the database schema.

Example: Assertion

- In Sells(bar, beer, price), no bar may charge an average of more than $5.

```
CREATE ASSERTION NoRipoffBars CHECK
   (
   NOT EXISTS (
      SELECT bar FROM Sells
      GROUP BY bar
      HAVING 5.00 < AVG(price)
   ));
```

Bars with an average price above $5

Example: Assertion

- In Drinkers(name, addr, phone) and Bars(name, addr, license), there cannot be more bars than drinkers.

```
CREATE ASSERTION FewBar CHECK (
  (SELECT COUNT(*) FROM Bars) <=
  (SELECT COUNT(*) FROM Drinkers)
);
```

Timing of Assertion Checks

- In principle, we must check every assertion after every modification to any relation of the database.
- A clever system can observe that only certain changes could cause a given assertion to be violated.
  - Example: No change to Beers can affect FewBar. Neither can an insertion to Drinkers.