NULLs
Database Modifications
Data Types
Updateable Views

NULL Values

- Tuples in 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 the **length** has some value, but we don't know what it is.
  - Inapplicable: e.g., the value of attribute *spouse* for an unmarried person.

Comparing NULL's to Values

- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.
- When any value is compared to 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).

Three-Valued Logic

- To understand how AND, OR, and NOT work in 3-valued logic, think of
  - TRUE = 1, FALSE = 0, and UNKNOWN = ½.
  - AND = MIN
  - OR = MAX
  - NOT(x) = 1-x

Example:

TRUE **AND** (FALSE **OR NOT**(UNKNOWN)) =

MIN(1, MAX(0, (1 - ½))) =

MIN(1, MAX(0, ½)) = MIN(1, ½) = ½.

Surprising Example

```
SELECT *
FROM Movies
WHERE length <=120 OR length > 120;
```

- Suppose that we have some NULL values in the length.
- What's the result?

We will get all the movies with a known length. Those with a length of NULL will not be in the result.

Checking for NULLs

- Can’t meaningfully use = or <>
- Should use:

  IS NULL

  IS NOT NULL

E.g.

```
SELECT *
FROM Movies
WHERE length IS NOT NULL;
```

NULL's Ignored in Aggregation

- **NULL** never contributes to a sum, average, or count, and can never be the minimum or maximum of a column.

```
SELECT SUM(length)
FROM Movies;
```

- But if there are **no non-NULL** values in a column, then the result of the aggregation is NULL.

Example: Effect of NULL's

```
SELECT count(*)
FROM Movies
WHERE studioName = 'Disney';
```

The number of movies from Disney.

```
SELECT count(length)
FROM Movies
WHERE studioName = 'Disney';
```

The number of movies from Disney with a known length.

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

- Consider **MovieExec**(name, address, cert#, netWorth)

```
INSERT INTO MovieExec
VALUES('Melanie Griffith', '34 Boston Blvd', 700, 300000);
```

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes.

```
INSERT INTO MovieExec(name, address, cert, netWorth)
VALUES('Melanie Griffith', NULL, 700, 3000000);
```

- 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.

Inserting Many Tuples

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

```
INSERT INTO <relation>
<query>;
```

Example

```
CREATE TABLE DisneyMovies(
    name VARCHAR2(25),
    year INT
);

INSERT INTO DisneyMovies
    SELECT title, year
    FROM Movie
    WHERE studioName = 'Disney';
```

Deletion

- To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>
WHERE <condition>;
```

Example

- Delete from the **Movie** table the Disney's movies:

```
DELETE FROM Movie
WHERE studioName ='Disney';
```

Example: Delete all Tuples

- Make the relation Movie empty:

```
DELETE FROM Movie;
```

- No WHERE clause needed here.

Updates

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

```
UPDATE <relation>
SET <list of attribute assignments>
WHERE <condition on tuples>;
```

Example

- Change the length of 'Godzilla' to 200.

```
UPDATE Movies
SET length = 200
WHERE title = 'Godzilla';
```

Another Example

- Suppose that Tom Cruise's movies have approximately 20 min of info before starting.
- So, let's take that 20 min off.

```
UPDATE Movies
SET length = length - 20
WHERE (title, year) IN
    (SELECT title, year
    FROM StarsIn
    WHERE starname = 'Tom Cruise');
```

Exercise

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

a) Using two INSERT statements, store in the database the fact that PC model 1100 is made by manufacturer C, has speed 1800, RAM 64g, hard disk 2t, and sells for $2000.

b) Insert the facts that for every PC there is a laptop with the same manufacturer, speed, RAM and hard disk, a 15-inch screen, a model number 1000 greater, and a price $500 more.

c) Delete all PC's with less than 500 GB of hard disk.

d) Delete all laptops made by a manufacturer that doesn't make printers.

e) Manufacturer A buys manufacturer B. Change all products made by B so they are now made by A.

f) For each PC, double the amount of RAM and add 1t to the amount of hard disk.

g) For each laptop made by manufacturer B, add one inch to the screen size and subtract $100 from the price.

Data Types

- Most common types are:
  - NUMBER(*m*) or NUMBER(*m,n*)

    [accepts *m* digits at all with *n* being after the decimal period]
  - INT or INTEGER (synonyms).
  - FLOAT(*p*)
    - [p is precision, from 1 to 126]
  - CHAR(*n*) = fixed-length string of *n* characters.
  - VARCHAR(*n*) = variable-length string of up to *n* characters.
  - DATE

Dates and Times

- DATE and TIME are types in SQL.
- No TIME type in ORACLE, but DATE also keeps the time.

```
CREATE TABLE Movies(
        title CHAR(20),
        year INT,
        length INT,
        studioName CHAR(20),
        release_date DATE,
        PRIMARY KEY (title, year)
);
```

Or

```
CREATE TABLE Movies(
        title CHAR(20),
        year INT,
        length INT,
        studioName CHAR(20),
        release_date DATE DEFAULT SYSDATE,
        PRIMARY KEY (title, year)
);
```

Getting a Date in/out

```
INSERT INTO Movies(title, year, length, studioName, release_date)
VALUES('Godzilla', 1998, 120, 'Paramount', '1998-02-12');

INSERT INTO Movies(title, year, length, studioName, release_date)
VALUES('Pretty Woman', 1990, 120, 'Touchstone', '13-09-90');
```

Error! Date not in proper format

```
INSERT INTO Movies(title, year, length, studioName, release_date)
VALUES('Pretty Woman', 1990, 120, 'Touchstone', TO_DATE('13-09-90', 'dd-mm-yy'));
```

Getting a Date in/out (II)

Getting the date and time out:

```
SELECT TO_CHAR(release_date, 'DD-MON-YYYY:HH:MI:SS')
FROM Movies;
```

For more info: http://www-db.stanford.edu/~ullman/fcdb/oracle/or-time.html

Adding/Deleting/Modifying Attributes

```
ALTER TABLE StarsIn ADD salary INT;

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

ALTER TABLE Movies MODIFY phone CHAR(18);

ALTER TABLE Movies DROP COLUMN phone;
```

Also in ORACLE:

```
ALTER TABLE StarsIn RENAME COLUMN title TO movieTitle;
```

Updateable Views - **WITH CHECK OPTION**

Only when:

1. There is only one relation, say R, in the FROM clause (of the query defining the view).

2. The list in the SELECT clause includes enough attributes that for every tuple inserted into the view, we can fill the other attributes out with NULL or the default, and have a tuple that will yield the inserted tuple in the view.

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movie
    WHERE studioName = 'Paramount'
WITH CHECK OPTION;

INSERT INTO ParamountMovie
VALUES ('Star Trek', 1979);
```

This insertion will fail!

Why this insertion is not possible?

The rationale for this behavior is:

- The above insertion, were it allowed to get through, would insert a tuple with NULL for studioName in the underlying Movie table.
- However, such a tuple doesn't satisfy the condition for being in the ParamountMovie view!
- Thus, it shouldn't be allowed to get into the database through the ParamountMovie view.

```
CREATE VIEW ParamountMovie2 AS
    SELECT studioName, title, year
    FROM Movie
    WHERE studioName = 'Paramount'
WITH CHECK OPTION;

INSERT INTO ParamountMovie2
VALUES ('Paramount', 'Star Trek', 1979);
```

Now it succeeds. Why?

Deleting

```
DELETE FROM ParamountMovie
WHERE year=2008;
```

is translated into

```
DELETE FROM Movie
WHERE year=2008 AND studioName='Paramount';
```

Updating

```
UPDATE ParamountMovie
SET year = 1979
WHERE title= 'Star Trek';
```

is equivalent to the base-table update

```
UPDATE Movies
SET year = 1979
WHERE title = 'Star Trek' AND
                    studioName = 'Paramount';
```

Top-n query in Oracle

E.g. **StarredIn** (celeb, movie)

Find the top-5 celebs in terms of movies they have starred in.

```
SELECT *
FROM
  (SELECT celeb, count(movie) AS cnt
   FROM StarredIn
   GROUP BY celeb
   ORDER BY count(movie) DESC)
WHERE rownum <= 5;
```