SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks,
  - Basic features, ordering, duplicates
  - Set operations (union, intersect, except)
  - Aggregation & Grouping
  - Nested queries (correlation)
  - Null values

NULLS in SQL

- Whenever we don’t have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute whether it can be null (*nullable* attribute)
- How does SQL cope with tables that have NULLs ?

Null Values

- If x= NULL then 4*(3-x)/7 is still NULL

- If x= NULL then x="Joe" is UNKNOWN
- In SQL there are three boolean values:

FALSE	=	0
UNKNOWN	=	0.5
TRUE	=	1

Null Values

- C1 AND C2 = min(C1, C2)
- C1 OR C2 = max(C1, C2)
- NOT C1 = 1 – C1

```
SELECT *
FROM Person
WHERE  (age < 25) AND
       (height > 6 OR weight > 190)
```

E.g.
age=20
heigth=NULL
weight=200

Rule in SQL: include only tuples that yield TRUE

Null Values

Unexpected behavior:

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

Some Persons are not included !

Null Values

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

Outer Joins

- In a typical join, tuples of one relation that don’t match any tuple on join conditions are omitted from result.
- In an outer join, tuples without a match may be preserved in the output.
- Missing values are filled with NULL.

Outer Joins

- LEFT OUTER JOIN: rows of **left** relation without matching row in right relation appear in result.
- RIGHT OUTER JOIN: rows of **right** relation without matching row in left relation appear in result.
- FULL OUTER JOIN: rows of both relations appear in result.

Outer Joins

Sailors

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55.5
58	rusty	10	35

Reserves

sid	bid	day
22	101	10/10
58	103	10/10

```
SELECT S.sid, R.bid
FROM Sailors S LEFT OUTER JOIN Reserves R
```

sid	bid
22	101
31	NULL
58	103

SQL Overview

- SQL Preliminaries
- Integrity constraints
- Query capabilities
  - SELECT-FROM-WHERE blocks,
  - Basic features, ordering, duplicates
  - Set ops (union, intersect, except)
  - Aggregation & Grouping
  - Nested queries (correlation)
  - Null values
- **Modifying the database**
- Views

Review in the textbook, Ch 5

Modifying the Database

Three kinds of modifications

- Insertion - creates new tuple(s)
- Deletion - remove existing tuple(s)
- Updates - modify existing tuple(s)

Sometimes they are all called “updates”

Insertions

General form:

```
INSERT INTO R(A1,…., An) VALUES (v1,…., vn)
```

Example: Insert a new sailor to the database:

```
INSERT INTO Sailor(sid, sname, rating, age)
            VALUES (3212, 'Fred', 9, 44)
```

Missing attribute → NULL.

May drop attribute names if give them in order.

Insertions

```
INSERT   INTO   Sailor(sname)

    SELECT  DISTINCT  B.name
    FROM      Boaters B
    WHERE   Boaters.rank = “captain”
```

The query replaces the VALUES keyword.
Here we insert *many* tuples into PRODUCT

Deletions

Example:

```
DELETE
FROM    Sailor
WHERE   S.sname = 'Horatio'
```

Factoid about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.

Updates

Example:

```
UPDATE   Sailor S
SET    rating = rating + 1
WHERE  Sailor.sid  IN
                (SELECT sid
                  FROM   Reserves R
                  WHERE  R.date ='Oct, 25');
```

Views

Views

- A **view** is a relation defined by a query.
- The query defining the view is called the **view definition**
- For example:

SQL

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = “Development”
```

Virtual and Materialized Views

- A view may be:
  - **virtual**: the view relation is defined, but not computed or stored.
    - Computed only on-demand – slow at runtime
    - Always up to date
  - **materialized**: the view relation is computed and stored in system.
    - Pre-computed offline – fast at runtime
    - May have stale data

Virtual view example

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, category)

```
CREATE VIEW  Seattle-view  AS

        SELECT  buyer, seller, product, store
        FROM    Person, Purchase
        WHERE   Person.city = "Seattle"    AND
                Person.name = Purchase.buyer
```

We have a new **virtual** table:
Seattle-view(buyer, seller, product, store)

View Example

We can use the view in a query as we would any other relation:

```
SELECT   name, store
FROM     Seattle-view, Product
WHERE    Seattle-view.product = Product.name  AND
         Product.category = “shoes”
```

Querying a virtual view

```
SELECT   name, Seattle-view.store
FROM     Seattle-view, Product
WHERE    Seattle-view.product = Product.name  AND
         Product.category = “shoes”
```

“View expansion”

```
SELECT  name, Purchase.store
FROM    Person, Purchase, Product
WHERE   Person.city = “Seattle”    AND
        Person.name = Purchase.buyer   AND
        Purchase.product = Product.name  AND
        Product.category = “shoes”
```

The great utility of views

- Data independence
- Efficient query processing
  - materializing certain results can improve query execution
- Controlling access
  - Grant access to views only to filter data
- Data integration
  - Combine data sources using views

View-related issues

1. View selection
   - which views to materialize, given workload
2. View maintenance
   - when base relations change, (materialized) views need to be refreshed.
3. Updating virtual views
   - can users update relations that don't exist?
4. Answering queries using views
   - when only views are available, what queries over base relations are answerable?

Materialized View Maintenance

- Two steps:
  - Propagate: Compute changes to view when data changes.
  - Refresh: Apply changes to the materialized view table.
- Maintenance policy: Controls when we do refresh.
  - Immediate: As part of the transaction that modifies the underlying data tables. (+ Materialized view is always consistent; - updates are slowed)
  - Deferred: Some time later, in a separate transaction. (- View becomes inconsistent; + can scale to maintain many views without slowing updates)

Deferred Maintenance

- Three flavors:
  - Lazy: Delay refresh until next query on view; then refresh before answering the query.
  - Periodic (Snapshot): Refresh periodically. Queries possibly answered using outdated version of view tuples. Widely used, especially for asynchronous replication in distributed databases, and for warehouse applications.
  - Event-based: E.g., Refresh after a fixed number of updates to underlying data tables.

Updating Virtual Views

How can I insert a tuple into a table that doesn't exist?

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = "Development"
```

If we make the following insertion:

```
INSERT INTO  Developers  VALUES("Joe", "Optimizer")
```

It becomes:

```
INSERT INTO  Employee(ssn, name, department, project, salary)
VALUES(NULL, "Joe", "Development", "Optimizer", NULL)
```

Non-Updatable Views

Person(name, city)
Purchase(buyer, seller, product, store)

```
CREATE VIEW  City-Store  AS

        SELECT Person.city, Purchase.store
        FROM     Person, Purchase
        WHERE  Person.name = Purchase.buyer
```

How can we add the following tuple to the view?

("Seattle", "Nine West")

We don't know the name of the person who made the purchase; cannot set to NULL.

Troublesome examples

```
CREATE VIEW  OldEmployees AS
  SELECT name, age
  FROM  Employee
  WHERE age > 30
```

```
INSERT INTO  OldEmployees  VALUES("Joe", 28)
```

If this tuple is inserted into view, it won't appear. Allowed by default in SQL!

Ambiguous updates

Name	group
Alice	fac
Bob	fac
Bob	cvs

group	file
fac	foo.txt
fac	bar.txt
cvs	foo.txt

Join →

view

Alice	foo.txt
Alice	bar.txt
Bob	foo.txt
Bob	bar.txt

Delete (“Alice”, “foo.txt”)

Updating views in practice

- Updates on views highly constrained:
  - SQL-92: updates only allowed on single-table views with projection, selection, no aggregates.
  - SQL-99: takes into account primary keys; updates on multiple table views may be allowed.
  - SQL-99: distinguishes between updatable and insertable views