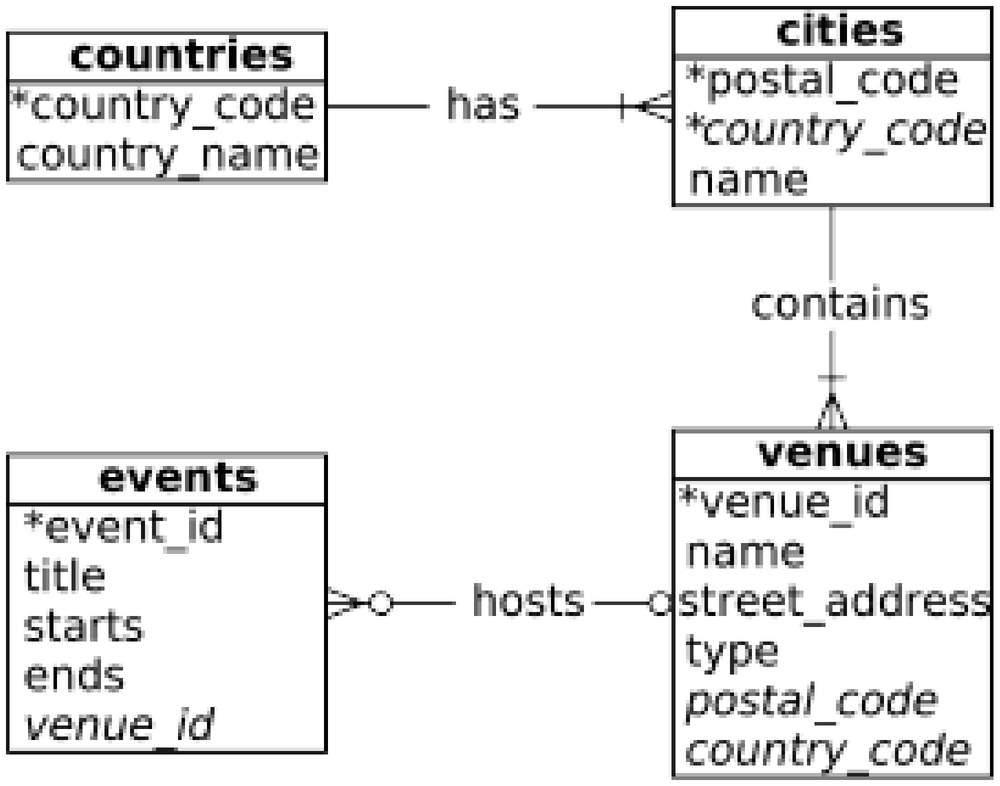
QUERIES and AGGREGATES

See http://sqlzoo.net/

Our working example



Let’s add more venues and events.

Basic SQL Query

Optional

Attributes from input relations

SELECT [DISTINCT] *target-list*

List of relations

FROM *relation-list*

WHERE *qualification*

Attr1 **op** Attr2

OPS: <, >, =, <=, >=, <>

Combine using AND, OR, NOT

- Semantics/Conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
- Not an efficient evaluation plan! (Optimzier picks efficient plans)

Find venues with events on ‘19-Oct-2019’

```
SELECT v.name as Venue
        FROM events e, venues v
        WHERE e.venue_id = v.venue_id
        AND (DATE(e.starts)  = '2019-10-19')
```

- Add DISTINCT to this query. Effect?

- Equivalent SQL using JOIN?

Expressions and Strings

```
SELECT e.title,
        to_char(e.starts, 'FMDay, Mon FMDD YYYY HH12:MI AM') as starts
        FROM events e
        WHERE e.title LIKE '%visit%'
        ORDER BY e.title;
```

- Illustrates date formatting and string pattern matching
- AS is a way to name fields in result
- LIKE is used for string matching. `_’ stands for any one character and `%’ stands for 0 or more arbitrary characters.

Find venues with events either in ’22-Apr-2020’ or ‘23-Apr-2020’

- UNION: Compute the union of two *union-compatible* sets of tuples
  - Same number/types of fields.
- Also available: INTERSECT and EXCEPT (What do we get if we replace UNION by EXCEPT?)
- SQL oddities: duplicates with union, except, intersect
  - Default: eliminate duplicates!
  - Use ALL to keep duplicates

```
SELECT v.name as Venue, v.country_code
    FROM events e, venues v
    WHERE e.venue_id = v.venue_id AND
    (DATE(e.starts) = '2020-04-22' OR
    DATE(e.starts) = '2020-04-23')
```

```
SELECT v.name as Venue, v.country_code
    FROM events e, venues v
    WHERE e.venue_id = v.venue_id
    AND DATE(e.starts) = '2020-04-22'
UNION
SELECT v.name as Venue, v.country_code
    FROM events e, venues v
    WHERE e.venue_id = v.venue_id
    AND DATE(e.starts) = '2020-04-23'
```

Find venues with events either in ’22-Apr-2020’ AND ‘23-Apr-2020’

- INTERSECT: Compute the intersection of any two *union-compatible* sets of tuples.
- In the SQL/92 standard, but some systems don’t support it.

```
SELECT v.name as Venue, v.country_code
    FROM events e1, events e2, venues v
    WHERE e1.venue_id = v.venue_id
    AND e2.venue_id = v.venue_id
    AND DATE(e1.starts) = '2020-04-22’
    AND DATE(e2.starts) = '2020-04-23';
```

```
SELECT v.name as Venue, v.country_code
    FROM events e, venues v
    WHERE e.venue_id = v.venue_id AND
    DATE(e.starts) = '2020-04-22'
INTERSECT
SELECT v.name as Venue, v.country_code
    FROM events e, venues v
    WHERE e.venue_id = v.venue_id AND
    DATE(e.starts) = '2020-04-23';
```

Aggregate Operators

```
COUNT (*)
COUNT ( [DISTINCT] A)
SUM ( [DISTINCT] A)
AVG ( [DISTINCT] A)
MAX (A)  Can use Distinct
MIN (A)  Can use Distinct
```

single column

```
SELECT count(title) FROM events
```

```
SELECT COUNT (DISTINCT title)
FROM events
```

```
SELECT min(starts), max(ends)
        FROM events INNER JOIN venues
        ON events.venue_id = venues.venue_id
        WHERE venues.name = 'Crystal Ballroom';
```

```
SELECT e.title FROM events e
    WHERE e.venue_id IN (SELECT v.venue_id FROM venues v
          WHERE v.name = 'University of Iowa') AND
    (e.ends-e.starts) = (SELECT max(e2.ends-e2.starts)
          FROM events e2 WHERE e2.venue_id IS NOT NULL);
```

Find date & title of the first event

- The first query is illegal! (wait for GROUP BY.)

```
SELECT e.title, min(e.starts)
FROM events e
```

How many tuples in the result?

```
SELECT e.title, e.starts FROM events e
    WHERE e.starts = (SELECT
    min(e2.starts) FROM events e2);
```

GROUP BY and HAVING

- Apply aggregate to each of several *groups* of tuples
- Find the number of events registered *for each venue*

For *each venue i*:

```
SELECT  COUNT (*)
FROM  events
WHERE  venue_id = i
```

```
SELECT venue_id, count(*)
        FROM events
        GROUP BY venue_id;
```

Queries With GROUP BY and HAVING

```
SELECT      [DISTINCT]  target-list
FROM           relation-list
WHERE         qualification
GROUP BY  grouping-list
HAVING group-qualification
```

How many tuples in the result?

- The target-list contains
  - Attribute names: must be a subset of *grouping-list*.
  - Terms with aggregate operations (e.g., COUNT (*)).
- The group-qualification
  - Must have a single value per group

Conceptual Evaluation

- Cross-product -> discard tuples -> apply projection -> partition into groups using the *grouping-list* attribute values -> eliminate groups that don't satisfy the *group-qualification*

- Expressions in *group-qualification* have a single value per group!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

Find the venues with at least 2 events in 2019

```
SELECT venue_id, count(*)
        FROM events
        WHERE venue_id IS NOT NULL AND
        EXTRACT(YEAR FROM starts)=2019
        GROUP BY venue_id
        HAVING count(*) >= 2;
```

Null Values

- Represent
  - *unknown* (e.g., rating not assigned) or
  - *inapplicable* (e.g., no spouse's name)
- Complications with nulls:
  - Operators to check if value is/is not *null*.
  - Is *rating > 8* true or false when *rating* is null?
    - Answer: Evaluate to unknown
  - What about AND, OR and NOT connectives?
    - Need 3-valued logic (true, false and *unknown*)
      - Not unknown = unknown
  - WHERE clause eliminates rows that don't evaluate to true
  - New operators (in particular, *outer joins*) possible/needed.

p	**q**	**p AND q**	**p OR q**
T	**T**	T	T
T	**F**	F	T
T	**U**	U	T
F	**T**	F	T
F	**F**	F	F
F	**U**	F	U
U	**T**	U	T
U	**F**	F	U
U	**U**	U	U

Window functions – Partition by

- Not all RDBMS support it
- Similar to GROUP BY
  - Calculates aggregates on the OVER a PARTITION of the result set.
  - Rather than grouping the results outside of the SELECT attribute list, it returns grouped values as any other field

```
SELECT venue_id, count(*)
        OVER (PARTITION BY venue_id)
        FROM events
        ORDER BY venue_id;
```

```
SELECT v.venue_id, v.name, e.title, count(*)
        OVER (PARTITION BY v.venue_id)
        FROM events e INNER JOIN venues v
        ON e.venue_id = v.venue_id;
```