Query language

Subhasis Ray

2023-02-01

The Relational Model

- "A relational model of data for large shared data banks" by E. F. Codd working at IBM (IBM was slow to adopt his work).
- Global DBMS market revenue in 2020 was 80 billion USD (`https://www.statista.com/`).

NOTE: these slides follow the textbook Database Systems by Connolly and Begg.

The Relational Model

- Relation: Table with columns and rows
- Attribute: Named column/field of a relation
- Domain: Set of allowable values for one or more attributes
- Tuple: Row/record in a relation
- Degree: Number of attributes in a relation
- Cardinality: Number of tuples in a relation

Mathematics of Relations

- Cartesian product of two sets A and B: the set of all ordered pairs (x, y) such that $x \epsilon A$ and $y \epsilon B$.
- Example: $A = \{2, 3\}$, $B = \{1, 5, 7\}$, then $A \times B = \{(2, 1), (2, 5), (2, 7), (3, 1), (3, 5), (3, 7)\}$
- Any subset of a Cartesian product is a relation.
- This is extended to any finite number of sets:
  $A \times B \times C = \{(x, y, z) | x \epsilon A, y \epsilon B, z \epsilon C\}$ and so on.
  $S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \cdots, x_n) | x_1 \epsilon S_1, x_2 \epsilon S_2, x_n \epsilon S_n\}$
  Written in short: $\prod_1^n S_i$
- The sets are domains, the ordered tuples are records/rows. In math, order matters, in database it does not (as long as it is the same for all rows).

Database Relations

- A relation schema - defined by a set of attribute and domain name pairs.
- A relational database is a set of tables, and a relational database schema is a set of relation schemas.

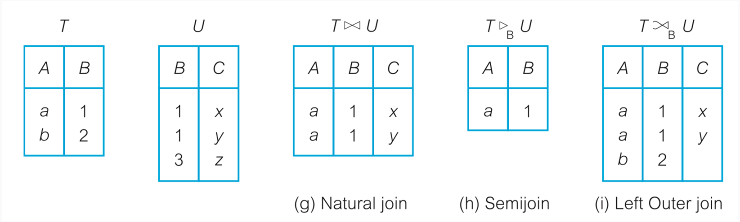
Keys

- Superkey: An attribute or set of attributes, that uniquely identifies a tuple within a relation.
- Candidate key: A minimal superkey (i.e., no proper subset of it can uniquely identify a tuple)
- Primary key: A candidate key that has been designated for uniquely identifying tuples within a relation.
- Foreign key: An attribute or set of attributes that matches a candidate key of some relation. This is used for matching rows across tables - indicates a relationship between the tables.
- View: a virtual relation that is derived dynamically from relations physically stored in a database.

A bit of relational algebra

- Selection $\sigma_{condition}(A)$: select rows based on condition (also called predicate)
- Projection $\Pi_{c_1, c_2, \ldots, c_n}(A)$: select columns $c_1, c_2, \cdots, c_n$
- Union $A \cup B$: put together the rows of two tables (they must have the same columns)
- Set difference $A - B$: rows that are in table A but not in B
- Intersection $A \cap B$: rows that are present in both tables
- Cartesian product $A \times B$: concatenation of every row in A with every row in B.
    - Question: How many attributes does it have? How many tuples?

Joins



T

A	B
a	1
b	2

U

B	C
1	x
1	y
3	z

$T \bowtie U$

A	B	C
a	1	x
a	1	y

(g) Natural join

$T \triangleright_B U$

A	B
a	1

(h) Semijoin

$T ⟕_B U$

A	B	C
a	1	x
a	1	y
b	2	

(i) Left Outer join

Credit: Database Systems by Connolly and Begg

- Θ join: rows from the Cartesian product that satisfy a condition, i.e. $\sigma_{condition}(A \times B)$, where the condition compares some field of A with some field of B. If the comparison is equality, then it is called *equijoin*.
- Natural join: equijoin of A and B on all common attributes.
- Left outer join: Where all the rows in the left table are kept, but rows for which there is no match on the right, the right attributes are made null. Similarly, right outer join and full outer join.

Aggregate functions and grouping operation

- Aggregate functions perform some kind of aggregation on the data: e.g., COUNT, SUM, AVG, MIN, MAX.
- Grouping operation groups rows by some attribute (and may then apply some aggregate functions on each such group)

Structured Query Language (SQL)

Aimed at being a natural english-like language for querying databases.

- Data Definition Language for defining database structure.

```
CREATE TABLE students (SID VARCHAR(8), FirstName VARCHAR(64),
↪  LastName VARCHAR(64));

CREATE TABLE marks (SID VARCHAR(8), Subject VARCHAR(4), Marks
↪  DECIMAL(4,2));
```

- Data Manipulation Language for retrieving and updating data.

```
INSERT INTO students VALUES ('U2021003', 'Peter', 'Norvig');

SELECT SID, Subject, Marks FROM marks WHERE Marks < 40;
```

SELECT

The most used SQL command.

```
SELECT
  [ALL | DISTINCT]
  col_expr [, col_expr] ...
  FROM table_references
  [WHERE where_cond]
  [GROUP BY {col_list}]
  [HAVING where_cond]
  [ORDER BY {col_list}]
```

```
SELECT SID, FirstName
↪   FROM students;
```

```
SELECT * FROM marks;
```

- Question: What could DISTINCT do?

SELECT with WHERE

- Conditions (<, >, , <>, >, <=, AND, OR, NOT), set membership

```
SELECT * FROM marks WHERE (Marks < 40) OR
↪  (Marks > 80);

SELECT * FROM marks WHERE Subject IN ('DMPR',
↪  'FOCS');

SELECT * FROM marks WHERE Subject NOT IN
↪  ('DMPR', 'FOCS');
```

SELECT with WHERE

- String pattern matching (LIKE, NOT LIKE)

```
SELECT * FROM Students WHERE FirstName LIKE
↪  'A%';

SELECT * FROM Students WHERE FirstName LIKE
↪  'Vi____';

SELECT * FROM Students WHERE FirstName NOT
↪  LIKE '%ish%';
```

Escape % with #.

SELECT with NULL

You cannot compare NULL to anything.

```
SELECT * FROM Students WHERE LastName IS NULL;
```

This is different from

```
SELECT * FROM Students WHERE LastName = '';
```

SELECT with ORDER BY

Use ORDER BY for sorting

```
SELECT * FROM Students ORDER BY LastName;
```

```
SELECT * FROM Students ORDER BY LastName DESC;
```

```
SELECT SID, LastName, FirstName FROM Students
↪   ORDER BY LastName, FirstName, SID;
```

SELECT with aggregate function

Aggregare functions ignore NULL values.

- Find the number of subjects

  ```
  SELECT COUNT(DISTINCT Subject) FROM marks;
  ```

Special case: COUNT(*) counts all rows regardless of NULL or duplicates.

- Total number of students (irrespective of NULL values anywhere)

  ```
  SELECT COUNT(DISTINCT Subject) FROM marks;
  ```

Grouping: SELECT with GROUP BY

- Each item in the SELECT list must have a single valued result per group.
- All columns in the SELECT list must appear in GROUP BY clause unless it is argument to an aggregate function.

```
SELECT SID, SUM(Marks) FROM marks GROUP BY
 ↪  SID ORDER BY SID;

SELECT SUM(Marks) FROM marks GROUP BY SID;
```

- Contrast with pandas `groupby`.

Restricting groups

HAVING clause

```
SELECT SID, SUM(Marks) FROM marks GROUP BY SID
↪  HAVING SUM(Marks) < 200;
```

Reference

- Database Systems: A Practical Approach to Design, Implementation, and Management by Thomas Connolly and Carolyn Begg