

# An application of FO: Relational Databases

Relations have named columns called attributes

```
frequents | drinker bar
-----|-----
|

serves    | bar beer
-----|-----
|

likes     | drinker beer
-----|-----
|
```

## 1 FO as a Query Language

Relational calculus is a variant of FO on relational vocabulary (no functions, just relations).

Here constants have a fixed interpretation – this is slightly different than in FO logic. For example, if “Joe” appears in a query, this can only be interpreted as “Joe”.

Examples of relational queries:

(i) Find all bars that serve Bud

$$\{b : bar | serves(b, Bud)\}$$

$b$  is a free variable, Bud is a constant.

(ii) Find the drinkers who frequent some bar that serves Bud

$$\{d : drinker | \exists b (frequents(d, b) \wedge serves(b, Bud))\}$$

(iii) Find the drinkers who frequent only bars serving Bud

$$\{d : drinker | \forall b [frequents(d, b) \longrightarrow serves(b, Bud)]\}$$

(iv) Find drinkers who frequent only bars serving some beer they like

$$\{d : drinker | \forall b (freq(d, b) \longrightarrow \exists c (serves(b, c) \wedge likes(d, c)))\}$$

## 2 SQL: Structured Query Language

SQL is the standard query language in relational databases. Its core is a syntactic variant of relational calculus. To see the flavor of the language, here is how queries (i)-(iii) can be expressed in SQL:

```
SELECT s.bar
FROM serves s
WHERE s.beer = 'Bud'
```

```
SELECT f.drinker
FROM freq f, serves s
WHERE f.bar = s.bar and s.beer = 'Bud'
```

```
SELECT drinker
FROM freq
WHERE dr NOT IN
(
  SELECT f.drinker
  FROM frequents f
  WHERE f.bar NOT IN
    (SELECT bar
     FROM serves
     WHERE beer = 'Bud')
)
```

## 3 Relational Algebra

This is a language equivalent to FO, consisting of simple operations on relations. Relational algebra is used in the implementation of SQL, as an intermediate representation language.

Main Operations

- Projection :  $\pi_X(R)$  projects  $R$  on a subset  $X$  of its columns (attributes).
- Selection :  $\sigma_{A \text{ op } B}(R)$  selects from  $R$  the subset of the tuples satisfying the condition  $A \text{ op } B$  where  $\text{op} \in \{=, \neq, \leq, \dots\}$ ,  $A$  and  $B$  are either attributes or constants, with at least one being an attribute.
- Set union and difference  $\cup, -$  : set operations applied to sets of tuples of the same arities.

- Join :  $R \bowtie P$  combines tuples from  $R$  and  $P$ , that agree on the common attributes. More precisely,  $R \bowtie P$  consists of the tuples  $t$  over  $att(R) \cup att(P)$  such that  $\pi_{att(R)}(t) \in R$  and  $\pi_{att(P)}(t) \in P$ .
- Attribute renaming:  $\delta_{A \rightarrow B}(R)$  renames attribute A to B in  $R$ .

Expressions built from these expressions are called a relational algebra queries. The algebra has the same expressive power as FO. This generalizes a classical result by Tarski. It was adapted by Ted Codd to the framework of relational databases.

Examples (queries (i)-(iii) expressed in the algebra):

(i)

$$\pi_{bar}(\sigma_{beer=Bud}(serves))$$

(ii)

$$\pi_{dr}(freq \bowtie \sigma_{beer=Bud}(serves))$$

(iii)

$$\pi_d(freq) - \pi_d[freq \bowtie (\pi_{bar}(freq) - \pi_{bar}(\sigma_{beer=Bud}(serves)))]$$

Relational algebra provides the basis for an efficient implementation of SQL. Stages in query processing: compilation of SQL into an algebra, logical query rewriting, and query evaluation plan generation. Use of indexes for efficient lookup of specified tuples in a relation. These techniques make SQL practical as a query language, even if data is very large. Note: complexity of FO is  $AC_0$ . This shows potential for efficient parallel processing of relational algebra queries.