Databases

Muchang Bahng

Fall 2024

Contents

This is a course on database languages (SQL), database systems (Postgres, SQL server, Oracle, MongoDB), and data analysis.

Definition 0.1 (Data Model)

A **data model** is a notation for describing data or information, consisting of 3 parts.
1. *Structure of the data.* The physical structure (e.g. arrays are contiguous bytes of memory or hashmaps use hashing). This is higher level than simple data structures.
2. *Operations on the data.* Usually anything that can be programmed, such as **querying** (operations that retrieve information), **modifying** (changing the database), or **adding/deleting**.
3. *Constraints on the data.* Describing what the limitations on the data can be.

There are two general types: relational databases, which are like tables, and semi-structured data models, which follow more of a tree or graph structure (e.g. JSON, XML).

1 Relational Databases

The most intuitive way to store data is with a *table*, which is called a relational data model, which is the norm since the 1990s.

Definition 1.1 (Relational Data Model)

A **relational data model** is a data model where its structure consists of
1. **relations**, which are two-dimensional tables.
2. Each relation has a set of **attributes**, or columns, which consists of a name and the data type (e.g. int, float, string, which must be primitive).[a]
3. Each relation is a set[b] of **tuples** (rows), which each tuple having a value for each attribute of the relation. Duplicate (agreeing on all attributes) tuples are not allowed.

So really, relations are tables, tuples are rows, attributes are columns.

Definition 1.2 (Schema)

The **schema** of a relational database just describes the form of the database, with the name of the database followed by the attributes and its types.

```
Beer (name string, brewer string)
Serves (bar string, price float)
...
```

Definition 1.3 (Instance)

The entire set of tuples for a relation is called an **instance** of that relation. If a database only keeps track of the instance now, the instance is called the **current instance**, and **temporal databases** also keep track of the history of its instances.

Finally, we talk about an important constraint.

[a] The attribute type cannot be a nonprimitive type, such as a list or a set.
[b] Note that since this is a set, the ordering of the rows doesn't matter , even though the output is always in some order.

Definition 1.4 (Key)

A set of attributes form a **key** for a relation if we do not allow two tuples in any relation instance to have the same values in all attributes of the key.

While we can make a key with a set of attributes, many databases use artificial keys such as unique ID numbers for safety.

SQL (Structured Query Language) is the standard query language supported by most DBMS. It is **declarative**, where the programmer specifies what answers a query should return,but not how the query should be executed. The DBMS picks the best execution strategy based on availability of indices, data/workload characteristics, etc. (i.e. provides physical data independence). It contrasts to a **procedural** or an **operational** language like C++ or Python. One thing to note is that keywords are usually written in uppercase by convention.

Definition 1.5 (Primitive Types)

The primitive types are listed.

1. *Characters.* `CHAR(n)` represents a string of fixed length n, where shorter strings are padded, and `VARCHAR(n)` is a string of variable length up to n, where an endmarker or string-length is used.
2. *Bit Strings.* `BIT(n)` represents bit strings of length n. `BIT VARYING(n)` represents variable length bit strings up to length n.
3. *Booleans.* `BOOLEAN` represents a boolean, which can be `TRUE`, `FALSE`, or `UNKNOWN`.
4. *Integers.* `INT` or `INTEGER` represents an integer.
5. *Floating points.* `FLOAT` or `REAL` represents a floating point number, with a higher precision obtained by `DOUBLE PRECISION`.
6. *Datetimes.* `DATE` types are of form `'YYYY-MM-DD'`, and `TIME` types are of form `'HH:MM:SS.AAAA'` on a 24-hour clock.

1.1 Tables, Attributes, and Keys

Before we can even query or modify relations, we should know how to make or delete one.

Theorem 1.1 (`CREATE TABLE`, `DROP TABLE`)

We can create and delete a relation using `CREATE TABLE` and `DROP TABLE` keywords and inputting the schema.

```
CREATE TABLE Movies(
  name CHAR(30),
  year INT,
  director VARCHAR(50),
  seen DATE
);

DROP TABLE Movies;
```

What if we want to add or delete another attribute? This is quite a major change.

Theorem 1.2 (`ALTER TABLE`)

We can add or drop attributes by using the `ALTER TABLE` keyword followed by
1. `ADD` and then the attribute name and then its type.
2. `DROP` and then the attribute name.

```
ALTER TABLE Movies ADD rating INT;
ALTER TABLE Movies DROP director;
```

Theorem 1.3 (`DEFAULT`)

We can also determine default values of each attribute with the `DEFAULT KEYWORD`.

```
ALTER TABLE Movies ADD rating INT 0;
...
CREATE TABLE Movies(
  name CHAR(30) DEFAULT 'UNKNOWN',
  year INT DEFAULT 0,
  director VARCHAR(50),
  seen DATE DEFAULT '0000-00-00'
);
```

Theorem 1.4 (`PRIMARY KEY`, `UNIQUE`)

There are multiple ways to identify keys.
1. Use the `PRIMARY KEY` keyword to make `name` the key. It can be substituted with `UNIQUE`.

```
CREATE TABLE Movies(
  name CHAR(30) PRIMARY KEY,
  year INT,
  director VARCHAR(50),
  seen DATE
);
```

2. Use the `PRIMARY KEY` keyword, which allows you to choose a combination of attributes as the key. It can be substituted with `UNIQUE`.

```
CREATE TABLE Movies(
  name CHAR(30),
  year INT,
  director VARCHAR(50),
  seen DATE,
  PRIMARY KEY (name, year)
);
```

1.2 Relational Algebra

We've talked about the structure of the data model, but we still have to talk about operations and constraints. We will focus on the operations here, which can be introduced with *relational algebra*, which gives a powerful way to construct new relations from given relations. Really, SQL is a syntactically sugared form of relational algebra.

The reason we need this specific query language dependent on relational algebra is that it is *less* powerful

than general purpose languages like C or Python. These things can all be stored in structs or arrays, but the simplicity allows the compiler to make huge efficiency improvements.

An algebra is really just an algebraic structure with a set of operands (elements) and operators.

Definition 1.6 (Relational Algebra)

A relational algebra consists of the following operands.
1. Relations R, with attributes A_i.
2. Operations.

It has the following operations.
1. *Set Operations.* Union, intersection, and difference.
2. *Removing.* Selection removes tuples and projection removes attributes.
3. *Combining.* Cartesian products, join operations.
4. *Renaming.* Doesn't affect the tuples, but changes the name of the attributes or the relation itself.

Let's take a look at each of these operations more carefully, using the following relation.

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

Figure 1: The example relation, which we will denote `serves`, which we will use to demonstrate the following operations.

Definition 1.7 (Set Operations)

Given relations R and S which must have the same schema (if not, just apply a projection), we can do the following set operations.
1. Union. $R \cup S$.
2. Intersection. $R \cap S$, which can be written also as $R - (R - S), S - (S - R)$, and surprisngly $R \bowtie S$.[a]
3. Difference. $R - S$.

Definition 1.8 (Selection)

The **selection** operator σ_p filters the tuples of a relation R by some condition p. It must be the case that p is deducible by looking only at that row.

$$\sigma_p R \tag{1}$$

This is analogous to the `WHERE` keyword.

```
SELECT *
FROM relation
WHERE
  p_1 AND p_2 AND ... ;
```

[a]The natural join will check for all attributes in each schema, but sine we assumed that they had the same schema, it must check for equality over all attributes.

Definition 1.9 (Projection)

The **projection** operator π_L filters the attributes of a relation R, where L is a subset of R's attributes.

$$\pi_L R \tag{2}$$

Note that since this operates on sets, if the projection results in two tuples mapping to the same projected tuple, then this repeated element is deleted. This is simply the `SELECT` keyword.

```
SELECT
  bar,
  beer
FROM beers;
```

Now let's talk about operations between two relations.

Definition 1.10 (Cartesian Product)

The **cartesian product** $S \times R$ of two relations is the relation

$$S \times R = \{(s \in S, r \in R)\} \tag{3}$$

which has a length of $|S| \times |R|$. It is commutative (so tuples are not ordered, despite its name), and if S and R have the same attribute name n, then we usually prefix it by the relation to distinguish it: $S.n, R.n$. In SQL, we can do it in one of two ways.

```
SELECT *
FROM table1
CROSS JOIN table2;

SELECT *
FROM table1, table2;
```

Definition 1.11 (Theta-Join)

The **theta-join** with **join condition/predicate** p gives

$$R \bowtie_p S = \sigma_p(R \times S) \tag{4}$$

1. If p consists of only equality conditions, then it is called an **equi-join**.
2. If p is not specified, i.e. we write $R \bowtie S$, called a **natural join**. The p is automatically implied to be
$$R.A = S.A \tag{5}$$
for all $A \in R.att \cap S.att$. Duplicate columns are always equal by definition and so one is removed, unlike equijoin, where duplicate columns are kept.

There are other types of joins that we will use.

Example 1.1 (Simple Filter)

Find all the addresses of the bars that Ben goes to.

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Table 1: Bar Information

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

Table 2: Frequents Information

We do the following.

$$\pi_{\text{address}}(\text{Bar} \bowtie_{\text{name=bar}} \sigma_{\text{drinker=Dan}}(\text{Frequents})) \tag{6}$$

Finally, we look at renaming.

Definition 1.12 (Renaming)

Given a relation R,

1. $\rho_S R$ means that you are chaning the relation name to S.
2. $\rho_{(A_1,\dots,A_n)} R$ renames the attribute names to $(A_1, \dots, A_n)$.
3. $\rho_{S(A_1,\dots,A_n)} R$ renames the relation name to S and the attribute names to $(A_1, \dots, A_n)$.

It does not really adding any processing power. It is only used for convenience.

1.3 Additional Operations

Definition 1.13 (Monotone Operators)

An operator $O(R, S, \dots)$ is monotone with respect to input R if increasing the size (number of rows/tuples) of R does not decrease the output relation O.

Example 1.2 ()

Every operator we went through is monotone, except for the set minus $R - S$, which is monotone w.r.t. R but not monotone w.r.t. S.