The Difference Between JCB™ and Previous Systems

Computers were invented because people want answers. Sometimes, the answers they want are the results of a complicated set of mathematical equations. Sometimes the answers they want can be found in a collection of documents. But often, people just have a question, and they want it answered. To these ends, various sorts of programs have been written.

The Options Today

When it comes to answering questions, the number of options has just increased. Here are the main alternatives:

- Most prominent in today's society are search engines. The two most famous are Google and Bing. In the case of search engines, the user types in one or more search terms, and a list of documents appears, ranked so that the documents most likely containing the information that the user wants appears at or near the top of the list. It is the user's job to read these documents, and find the question's answer.
- Another alternative is to use something like Watson. Such "expert" systems are very expensive, and are limited to a preset domain. For instance, Watson is famous for its ability to play Jeopardy. Such systems have also been set up for such domains as medical diagnostics.
- Sites such as Yahoo Answers allows you to ask a question, and get answers from real people. However, instead of these being professional answers, what you have on such systems is random people, self-selected, who show up and propose any answer to your question they like, or not.
- Specific programs can be written to answer specific types of problems.
- Last, and most important, we have systems that do Store-and-Query. Within this category, the most famous, and easiest to use, is SQL (standing for Structured Query Language). We now add to this list JCB (standing for James Cooke Brown), named after a researcher who believed (provably so) that spoken language could be formalized (meaning that pure rules could be applied). A happy by-product of a speakable, formal language is that people and computers can both be made to understand the language in a way not possible before.

Example 1: Presidents

When using a Store-and-Query language, the usual concept is that there are three steps involved.

- Step 1 is to decide how the data is to be stored, and define that storage lay-out so that the computer can understand the design we have in mind.
- Step 2 is to store the data, using that format.
- Step 3 is to issue a query, in order to retrieve the data.

Below, I'll use various alternatives to define the data (presidents' names, and terms in office), enter the data for the last 5 presidents (to keep the example reasonably short), and to ask the system who the president is. I'll do this SQL, APL, and JCB.

First, let's define the data in SQL. Here's what our user or database administrator would have to type (I'll use the IBM DB/2 dialect for these examples) to tell the system how we're setting up our data:

```
CREATE TABLE presidents(
   name     VARCHAR(64),
   starting TIMESTAMP,
   ending   TIMESTAMP);
```

Next, let's enter the data in SQL. Here's what our user or database administrator would have to type:

```
INSERT INTO presidents VALUES('Barack H. Obama',
   '2009-01-20 12:00:00.000000', '2017-01-20 12:00:00.000000');
INSERT INTO presidents VALUES('George W. Bush',
   '2001-01-20 12:00:00.000000', '2009-01-20 12:00:00.000000');
INSERT INTO presidents VALUES('William J. Clinton',
   '1993-01-20 12:00:00.000000', '2001-01-20 12:00:00.000000');
```

```
INSERT INTO presidents VALUES('George H. W. Bush',
   '1989-01-20 12:00:00.000000', '1993-01-20 12:00:00.000000');
INSERT INTO presidents VALUES('Ronald W. Reagan',
   '1981-01-20 12:00:00.000000', '1989-01-20 12:00:00.000000');
COMMIT;
```

Finally, let's use SQL to ask who the president is right now:

```
SELECT name FROM presidents
   WHERE CURRENT TIMESTAMP BETWEEN starting AND ending;
```

In JCB, we skip the design phase, and just go on to data entry:

> Beginning noon, January 20, 2009, ending noon, January, 2017, "Barack H. Obama" is president.
> Also beginning noon, January 20, 2001, ending noon, January, 2009, "George W.Bush" is president.
> Also beginning noon, January 20, 1993, ending noon, January, 2001, "William J. Clinton" is president.
> Also beginning noon, January 20, 1989, ending noon, January, 1993, "George H. W. Bush" is president.
> Also beginning noon, January 20, 1981, ending noon, January, 1989, "Ronald W. Reagan" is president.

Here, we use JCB to ask who the president is right now:

> Now, who is president?

<u>Example 2: Attributes</u>
Here's another store-and-query scenario. Let's say that I want to store certain information, attributes, actually, about some items. Let's also say that I want to categorize the information.

For this example, I'll have Fluffy, a female cat, Fido, a male dog, Nemo, a fish, and Excalibur, a sword. I also want to know whether something is animal, mineral, or vegetable. I'll ask who the dog is, whether Fluffy is female, and whether Nemo is an animal.

As usual, in SQL, we need to design our data:

```
CREATE TABLE attributes(
   name VARCHAR(32),
   attribute VARCHAR(32));

CREATE TABLE categories(
   attribute VARCHAR(32),
   category VARCHAR(32));
```

Now, let's fill in the basic data about the world:

```
INSERT INTO categories VALUES('cat', 'animal');
INSERT INTO categories VALUES('dog', 'animal');
INSERT INTO categories VALUES('fish', 'animal');
INSERT INTO categories VALUES('sword', 'mineral');
COMMIT;
```

Now, let's fill in the specific data:

```
INSERT INTO attributes VALUES('Fluffy', 'cat');
INSERT INTO attributes VALUES('Fido', 'dog');
INSERT INTO attributes VALUES('Nemo', 'fish');
INSERT INTO attributes VALUES('Excalibur', 'sword');
INSERT INTO attributes VALUES('Fluffy', 'female');
INSERT INTO attributes VALUES('Fido', 'male');
COMMIT;
```

Finally, we'll use SQL to ask the questions. First, who the dog is:

```
SELECT name FROM attributes WHERE attribute='dog';
```

Next, I'll ask whether Fluffy is female:

```
SELECT CASE WHEN COUNT(*)>0 THEN 'Yes' ELSE 'I don''t know' END
   FROM attributes
   WHERE name='Fluffy' AND attribute='Female';
```

Finally, I'll ask whether Nemo is an animal:

```
SELECT CASE WHEN COUNT(*)>0 THEN 'Yes' ELSE 'I don''t know' END
   FROM attributes, categories
   WHERE attributes.name='Nemo' AND
     attributes.attribute=categories.attribute AND
      categories.category='animal';
```

Now, let's reexamine the same problem in JCB: There's no data design to do, so let's get into entering some basic data about the world, namely definitions of sets of things:

Set animal cat; Execute.
Set animal dog; Execute.
Set animal fish; Execute.
Set mineral sword.

Now, we can enter our specific data:

"Fluffy" is a cat.
Also "Fido" is a dog.
Also "Nemo" is a fish.
Also "Fluffy" is female.
Also "Fido" is male.

Now we're ready to issue our queries. I'll list the three JCB queries here without bothering to describe what they do:

Who is a dog?

Is "Fluffy" female?

Is "Nemo" an animal?

Example 3: What about "No"?

You may or may not have noticed from the SQL that the only two possible answers from the questions were "Yes" and "I don't know". That's because SQL either finds something or it doesn't. If we add some programming to SQL, we can determine the "No" possibility. Let's add the capability to answer questions as to whether Fluffy is a dog, or whether fluffy is male.

In SQL, we've already got our data definitions. We need to add a bit more data about the world, though:

```
INSERT INTO categories VALUES('male', 'gender');
INSERT INTO categories VALUES('female', 'gender');
COMMIT;
```

Now, we can ask the questions. First, whether Fluffy is a dog:

```
SELECT CASE
   WHEN SUM(CASE WHEN attributes.attribute='dog' THEN 1 ELSE 0 END)>=1 THEN
      'Yes'
   WHEN SUM(CASE WHEN categories.attribute IS NOT NULL THEN 1 ELSE 0 END)>=1
      THEN 'No'
   ELSE 'I don''t know' END
FROM attributes
LEFT JOIN categoris ON
   attributes.attribute=categories.attribute AND
   categories.category='animal'
WHERE attributes.name='Fluffy'
```

To ask whether Fluffy is male, we have almost the same query:

```
SELECT CASE
   WHEN SUM(CASE WHEN attributes.attribute='male' THEN 1 ELSE 0 END)>=1 THEN
      'Yes'
   WHEN SUM(CASE WHEN categories.attribute IS NOT NULL THEN 1 ELSE 0 END)>=1
      THEN 'No'
   ELSE 'I don''t know' END
FROM attributes
LEFT JOIN categoris ON
   attributes.attribute=categories.attribute AND
   categories.category='gender'
WHERE attributes.name='Fluffy'
```

Next, let's go back to JCB. Likewise, in JCB, we need to give extra info about the world:

> Set gender male; Execute.
> Set gender female.

Now, we're ready to ask our two questions using JCB:

> Is "Fluffy" a dog?

> Is "Fluffy" male?

Example 4: Differences of opinion

In all existing data-base and knowledge-base systems, you have three choices for trust:

- You can trust someone to enter data, and to read the data, in which case, you can share the data equally,
- You can allow someone to read the data, without allowing them to write into the data, or
- You can set up a separate area, so that your data and their data are not visible to each other.

But, what if you trust someone to a given extent. For instance, you might want everyone to be able to enter data such that you can all see it, but you want to assign various levels of trust, so that some people's word is truth, while other people's word is valuable, and other people's word is opinion only, while others can enter data for themselves, but that data is not to be used or trusted by others.

Here's a very basic situation that can be handled by JCB:

- Bob states that liver is tasty.
- Carol states that liver is not tasty.

Those two facts contradict. In a normal knowledge system, either liver is tasty, or it's not. In JCB, however, if neither speaks with absolute authority, then both are opinions. If one has more authority or trust than the other, that level of trust can affect answers based on these opinions.

Actions

With the latest version, JCB is no longer restricted to storing and retrieving information. It can now be integrated with robotic systems, and act as an intelligent planning system. A robotic control system can control one or more robots. By making a simple request, any available body can be used:

> Please cook an egg.

Or, by calling out a name, a specific robotic body can be commanded:

> "Hewey" must cook an egg.

What if?

We've all seen movies in which robots get out of control. Isaac Asimov solved this for us with his "3 laws of robotics". Asimov's rule 1, for instance, is "A robot, through action or inaction, may not allow a human being to come to harm." JCB allows contravening principals, which can prevent unwanted actions from occurring.