Relational Database Languages

Tuple relational calculus
- ALPHA (Codd, 1970s)
- QUEL (based on ALPHA)
- Datalog (rule-based, like PROLOG)

Domain relational calculus
- QBE (used in Access)

History of SQL

Standards:
- SQL-86 (ANSI, ISO)
- SQL-89
- SQL-92 (SQL2)
- SQL:1999 (SQL3)
- SQL:2003
- SQL:2006
- SQL:2008
- SQL:2011

IBM developed SEQUEL in early 70s (Structured English Query Language)
Renamed SQL (Structured Query Language)

Many different flavors of SQL:
- sqlplus, SQLServer, MySQL, etc.

Here: Oracle SQL
Parts of SQL

**DDL**
Creating (CREATE),
Modifying (ALTER), and
Removing (DROP)

**DML**
Retrieving (SELECT),
Inserting (INSERT),
Modifying (UPDATE), and
Removing (DELETE).

**DCL**
Grant privileges (GRANT)
Revoking privileges (REVOKE)

---

**SELECT**

Combination of select and project operations.

Basic Syntax:

```
SELECT attribute_list
FROM table_list
WHERE condition
GROUP BY attribute_list
HAVING condition
ORDER BY attribute_list;
```

Example:

```
SELECT LastName, SID
FROM student
WHERE career = 'UGRD' AND started < 2008;
```

---

**SELECT * and empty WHERE**

```
SELECT *
FROM student;

SELECT *
FROM student, studentgroup;

SELECT *
FROM student, studentgroup
WHERE presidentID = SID
```
SELECT Examples

University
- List the SSNs of all students
- List all names of all student groups
- List the names of all programs
- List the name of all students in the IT department

Duplicates in SQL

Duplicates can occur if key attributes are not selected
SQL keeps duplicates, for several reasons:
- Cheaper to implement
  (duplicate elimination is expensive)
- Duplicates might be required
  (e.g., aggregate functions)

Removal of duplicates can be forced using DISTINCT

```
SELECT DISTINCT
```

Example:
- List cities in which students live (university).

Excursion:
How to write SQL Queries
How to Write Simple SQL Query

3 Stages
Before you write the SQL
Writing the SQL
After Writing the SQL

Before you write the SQL

Make sure you understand the problem.
Clarify if necessary
Do it by hand.
Reflect on what you did.

Writing the SQL

Start with FROM
which tables are involved, how often?

Then do WHERE
first join all tables (n tables need n-1 equals) - test
then add particular conditions

Finally, do SELECT
What info do you need to display
After Writing the SQL

Test
Run the query
Compare output to what you expected
sanity check: does it make sense?
In case of problems: read query

End of Excursion

Renaming (Aliases)

If necessary we can rename attributes:

SELECT LastName, ' ' || FirstName AS Name
FROM student;

and tables (drop AS in Oracle for table renaming)

SELECT S.Lastname, SG.Name
FROM student AS S, studentgroup AS SG
WHERE S.SID = SG.PresidentID;
Renaming Examples

University
• List the names of all students and expected graduation year
• List the names of student groups and the names of their presidents

Conditions

=   Equality
<   Less than
\leq   Less than or equal
\geq   Greater than
\neq or !=   not equal (depends on system)
LIKE   Allows Wildcards (Oracle)
\%   (any number of characters)
_   (single character)

Also (Oracle): REGEXP_LIKE (text, pattern)

Operators

+   Addition (works for dates in Oracle/Access)
-   Subtraction (works for dates in Oracle/Access)
*   Multiplication
/   Division
||   Concatenation (for strings)
and, or, not   Boolean operations
Functions (Oracle)

Strings and Numbers

String type:
- Length(s) Length of string s
- Rtrim(s), Ltrim(s) Delete trailing (leading) spaces

Numeric type:
- Round(x) Round x to integer
- Round(x,k) Round x to k positions
- Abs(x) Absolute value of x
- Exp(x) e^x

See http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/functions.htm#i1482196

Functions (Oracle)

Date/Timestamp

- extract date to integers
- last_day last day of month
- months_between months between two dates
- to_char(date, fmt) format date (e.g. MM, MON, MONTH)

SELECT extract (year from current_date) FROM dual;

Between and Ordering

For number types, ranges can be defined using BETWEEN.

SELECT LastName, SID
FROM student
WHERE started between 2011 and 2013;

The ORDER BY clauses allows ordered output
multiple attributes, expressions allowed
ASC (default) and DESC to specify order

SELECT LastName, SID
FROM student
ORDER BY started ;

SELECT LastName, SID
FROM student
ORDER BY started DESC, LastName;
Truth and Tables


\[ \begin{array}{c|c|}
\text{P} & \text{Q} \\
\hline
0 & 0 \\
0 & 1 \\
1 & 0 \\
1 & 1 \\
\end{array} \]

Definition: A function is a \textit{truth-function} iff its truth only depends on the truth of its arguments.

Theorem (Post): Every truth-function can be written using only negation and disjunction. (Or negation and conjunction.)

Logic examples I

• Students in computer science and information systems
• Graduate students unless they are PhDs.
• List all graduate students in computer science, computer graphics and artificial intelligence.
• List graduate and undergraduate students not from Chicago that started before 2010 and after 2012.
• List courses that violate the rule: All gaming courses must be 300 or above, all computer science courses must be 400 or above and IT courses must be at most 200 level.
• Check whether there are students that violate the following rules: Computer gaming students must be undergraduates, information system students must be graduates.
• List students that violate the rule: A PhD student cannot be an undergraduate.

Logic examples II

• List students that have both a graduate and an undergraduate record.
• List courses that have both grad and undergrad versions (go by title).
• List students that don’t have a SSN listed.
• List years in which no undergraduate computer science student started.
Logic examples II

• List students that have both a graduate and an undergraduate record.
• List courses that have both grad and undergrad versions (go by title).
• List students that don’t have a SSN listed.
• List years in which no undergraduate computer science student started.

Some of these we can’t do yet (4), for some we need additional knowledge (3), and some we can do (1,2) but there must be better solutions.

Nulls

Reasons:
• we don’t know value
• value isn’t applicable
• we don’t know whether value is applicable

Null is not a value (well), to test for it use
is null
is not null

Examples:
• List students that don’t have a SSN listed.
• List student groups that don’t currently have a president.
• List students that do have a SSN listed.
• We require that students that don’t have a last name do enter a first name. List students that violate this requirement.

Nulls in Conditions

Example

SELECT * FROM student WHERE SSN = 123123123;
SELECT * FROM student WHERE NOT(SSN = 123123123);
### Three-valued Logic

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p OR q</th>
<th>p AND q</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>Unknown</td>
</tr>
<tr>
<td>True</td>
<td>Unknown</td>
<td>True</td>
<td>Unknown</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>Unknown</td>
<td>True</td>
<td>True</td>
<td>Unknown</td>
</tr>
<tr>
<td>Unknown</td>
<td>False</td>
<td>True</td>
<td>Unknown</td>
</tr>
<tr>
<td>Unknown</td>
<td>Unknown</td>
<td>True</td>
<td>Unknown</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>Unknown</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

\[ p \lor q \begin{array}{c} \text{True} \\ \text{Unknown} \\ \text{False} \end{array} \begin{array}{c} \text{True} \\ \text{Unknown} \\ \text{False} \end{array} \begin{array}{c} \text{True} \\ \text{Unknown} \\ \text{False} \end{array} \begin{array}{c} \text{True} \\ \text{Unknown} \\ \text{False} \end{array} \]

\[ p \land q \begin{array}{c} \text{True} \\ \text{Unknown} \\ \text{False} \end{array} \begin{array}{c} \text{True} \\ \text{Unknown} \\ \text{False} \end{array} \begin{array}{c} \text{True} \\ \text{Unknown} \\ \text{False} \end{array} \begin{array}{c} \text{True} \\ \text{Unknown} \\ \text{False} \end{array} \]

**WHERE conditions:** have to be true  
**CHECK constraints:** can’t be false

\[ \neg p \begin{array}{c} \text{True} \\ \text{False} \\ \text{Unknown} \end{array} \begin{array}{c} \text{False} \\ \text{True} \\ \text{Unknown} \end{array} \begin{array}{c} \text{False} \\ \text{Unknown} \\ \text{False} \end{array} \begin{array}{c} \text{False} \\ \text{Unknown} \\ \text{False} \end{array} \]

### Nulls in Expressions

**Null in Operations**

\[ \text{null} \circ x = \text{null} \quad (\text{operation like } +, -, \ast, /, \text{etc.}) \]

SELECT EmpID, salary \ast (1 + \text{level} \ast .05)  
FROM employee;

**Null in Functions**

\[ f(\ldots, \text{null}, \ldots) = \text{null} \quad (\text{for most functions } f, \text{not all, e.g. } \text{||}) \]

SELECT \text{least}(\text{price\_new}, \text{price\_used})  
FROM bookwprices;

**Solution:** `\text{coalesce}(x, y, z, \ldots)` evaluates to first non-null value

### Objections to Nulls

- we don’t know what the null value represents  
  - Codd suggested two values:  
    - Missing Applicable/Missing Inapplicable
  - 3-valued behavior is counterintuitive and leads to SQL programs behaving incorrectly in presence of nulls
  - wastes space (needs extra indicator bit)

E.g., see the Third Manifesto: [http://en.wikipedia.org/wiki/The_Third_Manifesto](http://en.wikipedia.org/wiki/The_Third_Manifesto)
Date’s Example

Here I can present my argument. The fundamental point I want to make is that certain Boolean expressions—and therefore certain queries in particular—produce results that are correct according to three-valued logic but not always in the real world. As an example, consider the (nonrelational) database shown in Figure 4-2, in which the CITY is null for part P1. Note carefully that the empty slots in that figure, in the places where the CITY value for P1 should be, are not the same as the nulls in Figure 2-4, which are in the places where the .null values for P1 should be.

Figure 4-2: A nonrelational database, with a null

As I pointed out in Chapter 2, most relational databases are capable of coping with nulls, that is, nonexistent values. A relational database is designed to cope with nulls, but not necessarily to cope with empty slots.

Figure 2-4: A relational database, with an explicit empty slot

Table 2-1: The company P1 has a supplier, S1, that provides parts.

<table>
<thead>
<tr>
<th>Supplier</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>London</td>
</tr>
<tr>
<td>P1</td>
<td>PL</td>
</tr>
</tbody>
</table>

Consider now the following (unintended) rather contrived query on the database of Figure 4-2:

```
SELECT DISTINCT S, P, C
WHERE S.CITY = P.CITY
```

This query has only one row, since the CITY values of S1 and P1 are both null. This is a perfectly correct result. However, it is not what one would expect—hence the significance of the example. It is a good illustration of the need for a logic that can cope with empty slots and the various conflict points that arise from such slots.

From Chris Date, SQL and Relational Theory, O'Reilly, 2009.