Basic Predicates

Tests each record's entry for the given attribute against the given value
- May be true, false or unknown

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equality</td>
</tr>
<tr>
<td>&lt;, &lt;=</td>
<td>Less than (or equal to)</td>
</tr>
<tr>
<td>&gt;, &gt;=</td>
<td>Greater than (or equal to)</td>
</tr>
<tr>
<td>&amp;&amp;,</td>
<td></td>
</tr>
</tbody>
</table>
Working With Records

**INSERT INTO** `tableName` [(`attr1`, `attr2`, ...)]
**VALUES** (`value1`, `value2`, ...);

- Adds a new record with the given values to the table
  - Optional attribute list allows you to specify fewer fields

**UPDATE** `tableName`
**SET** `attr1` = `value1`, `attr2` = `value2`, ...
**WHERE** `predicate(s)`;

- Alters the records for which the `predicate(s)` are true

**DELETE FROM** `tableName`
**WHERE** `predicate(s)`;

- Deletes the records for which the `predicate(s)` are true

More Predicate Operators

**attribute BETWEEN** `value1` AND `value2`

- True for all records whose field is within the range
  `[value1, value2]` (inclusive)

**attribute IN** (`value1`, `value2`, ...)

- True for all records whose field is in the given set

**attribute LIKE** `stringPattern`

- True for all records whose field matches the given pattern
  - The field must be a string type
  - `stringPattern` is composed of characters and wildcards
    - `%` matches one or more characters
    - `_` (underscore) matches exactly one character
Some Useful Things

- MySQL is occasionally case-sensitive
  - Database and table names may be case-sensitive
  - Depends on the server
  - Typical convention
    - MySQL keywords are written in uppercase
    - Databases, tables and fields are named in lowercase

- Predicates can combined using **AND** and/or **OR**
  - Enclose in parentheses to ensure proper evaluation
    - e.g., `(pred1 AND pred2) OR pred3`

Searching

```
SELECT attr1, attr2, ...
FROM tableName
WHERE predicate(s);
```

- Selects records that satisfy the given `predicate(s)`
  - Selects only the attributes requested
    - The wildcard `*` can be used to specify all columns

- Use `SELECT DISTINCT` to display unique values
  - A subset of attributes may result in duplicate rows
Keys

- Completely unique values within each record
  - Composed of one or more attributes
  - No two records have the same key

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>show id</th>
<th>char id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Sherlock</td>
<td>10</td>
<td>221</td>
</tr>
<tr>
<td>47</td>
<td>Jane Eyre</td>
<td>10</td>
<td>222</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>221</td>
<td>Sherlock Holmes</td>
</tr>
<tr>
<td>222</td>
<td>Dr. Watson</td>
</tr>
</tbody>
</table>

Keys

- Keys should never be null
  - (And perhaps other values too, depending on the data)
- Each table really should have its key(s) specified

```
CREATE TABLE tableName(
  attr1 type1 NOT NULL,
  ...
  PRIMARY KEY(attr1, ...)
);

ALTER TABLE tableName
  MODIFY attr1 type1 const1 NOT NULL,
  ...
  ADD PRIMARY KEY(attr1, ...)
;
```
Joining Tables

**tableName1 JOIN tableName2**

- Creates a new temporary table
  - Contains each attribute from `tableName1` followed by each attribute in `tableName2`
  - Contains all possible pairs of records from `tableName1` and `tableName2`
    - Cartesian product

- Used in combination with **SELECT**
  - Provides ability to perform multi-table queries ...
    - with a more sophisticated form of **JOIN**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Sherlock</td>
</tr>
<tr>
<td>47</td>
<td>Jane Eyre</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>show_id</th>
<th>char_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>221</td>
</tr>
<tr>
<td>10</td>
<td>222</td>
</tr>
</tbody>
</table>

SELECT * FROM `shows` JOIN `dramatis_personae`;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>show_id</th>
<th>char_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Sherlock</td>
<td>10</td>
<td>221</td>
</tr>
<tr>
<td>10</td>
<td>Sherlock</td>
<td>10</td>
<td>222</td>
</tr>
<tr>
<td>47</td>
<td>Jane Eyre</td>
<td>10</td>
<td>221</td>
</tr>
<tr>
<td>47</td>
<td>Jane Eyre</td>
<td>10</td>
<td>222</td>
</tr>
</tbody>
</table>
Joining Tables

$tableName1$ JOIN $tableName2$ ON condition(s)

- Use condition(s) to combine records
  - $tableName1$.attr1 operator $tableName2$.attr2
  - All pairs of records with satisfactory fields are combined
  - All records that don't belong to a pair are dropped
- For best results, match the keys in the conditions
  - All records related to the same item are combined
- JOIN can be extended to any number of tables
  - $table1$ JOIN $table2$ ON cnd2 JOIN $table3$ ON cnd3 ...

### Example

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>show_id</th>
<th>char_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Sherlock</td>
<td>10</td>
<td>221</td>
</tr>
<tr>
<td>47</td>
<td>Jane Eyre</td>
<td>47</td>
<td>222</td>
</tr>
</tbody>
</table>

CISC 282 | Mar. 11th | MySQL: More Syntax

```sql
SELECT * FROM shows
JOIN dramatis_personae ON shows.id = dramatis_personae.show_id
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>show_id</th>
<th>char_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Sherlock</td>
<td>10</td>
<td>221</td>
</tr>
<tr>
<td>10</td>
<td>Sherlock</td>
<td>10</td>
<td>222</td>
</tr>
</tbody>
</table>

CISC 282 | Mar. 11th | MySQL: More Syntax
Joining Tables

```
SELECT * FROM shows JOIN dramatis_personae ON shows.id = dramatis_personae.show_id
JOIN characters ON dramatis_personae.char_id = characters.id;
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>show_id</th>
<th>char_id</th>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Sherlock</td>
<td>10</td>
<td>221</td>
<td>221</td>
<td>Sherlock Holmes</td>
</tr>
<tr>
<td>10</td>
<td>Sherlock</td>
<td>10</td>
<td>222</td>
<td>222</td>
<td>Dr. Watson</td>
</tr>
</tbody>
</table>

CISC 282 | Mar. 11th | MySQL: More Syntax