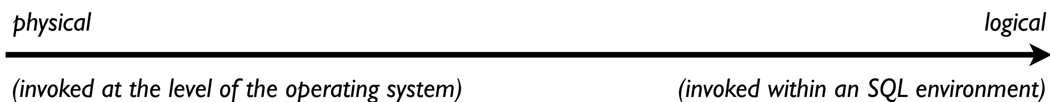
SQL

- Stands for "Structured Query Language"
- Essentially a "friendlier" (for both users and computers) version of the relational formalisms — algebra (which we've seen) and calculus (which we haven't seen yet)
- First developed by IBM under the name "Sequel" for its System R product
- Many generations of ANSI standard versions at this point: SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003

- SQL encompasses programmatic control of most of the functions expected of databases today:
  - Data definition language (DDL)
    - Data integrity
    - Views
  - Data manipulation language (DML)
  - Transaction control
  - Authorization and security
  - Access via general-purpose programming languages (embedded SQL, dynamic SQL, ODBC, JDBC)
- PostgreSQL "strongly conforms" (in the developers' words) to the ANSI SQL-92 and SQL:1999 standards
- One of the main distinctions between PostgreSQL and MySQL is its standards compliance (two words: *nested queries*) — tradeoff power and flexibility for speed

SQL and PostgreSQL

- Remember that SQL is a language *within* the relational data model — it operates on relational data model constructs using relational operations
- PostgreSQL is a relational database management *system* — it implements the relational data model, but as an implementation it involves many additional elements behind the scenes, all in service of providing us with as close an environment as possible to the "pure" realm of the theory behind relations
- SKS uses *logical* vs. *physical* to characterize this



initdb: Creates the physical database repository (or "cluster," in PostgreSQL terms)

createuser/dropuser: Creates/destroys a database user

createdb/dropdb: Creates/destroys a logical database

psql: Enters a command-line SQL environment that is "connected" to some logical database as some database user

programming interfaces (embedded SQL, dynamic SQL, ODBC, JDBC, etc.): Provides access to the SQL environment from assorted general-purpose programming languages and platforms

SQL statements: Performs relational database activity within a logical database — relations (tables), modification (insert/update/delete), queries/retrieval, transaction management, and many more

Data Definition Highlights

- *create table*: Creates relations, defines their attributes and domains, specifies primary and foreign keys
  - Built-in attribute domains (data types): *bit*, *boolean*, *char*, *varchar*, *int*, *smallint*, *numeric*, *real*, *double precision*, *float*, *date*, *time*, *timestamp*…and there are more that are specific to PostgreSQL (*bigint*, *serial*, *inet*, *money*)
- *drop table*: Removes relations from the logical database
- *alter table*: Modifies the structure of a relation (relation names, attribute names/types, constraints)

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name (
  { column_name data_type [ DEFAULT default_expr ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE parent_table [ { INCLUDING | EXCLUDING } DEFAULTS ] }  [, ... ]
)
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  UNIQUE [ USING INDEX TABLESPACE tablespace ] |
  PRIMARY KEY [ USING INDEX TABLESPACE tablespace ] |
  CHECK (expression) |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
  PRIMARY KEY ( column_name [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
  CHECK ( expression ) |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- relation: name, schema
- attributes: name, domain
- keys: primary, foreign

Data Manipulation: Modifications

These commands correspond to the database modification operations — the named table is the target of relational assignment, and the *where/values/select* clauses correspond to some relational expression

- *delete from…where*: Performs tuple deletion
- *insert into…values* or *insert into…select*: Performs tuple insertion (literal/constant or derived from query)
- *update…set…where*: Performs tuple modificatioin

```
DELETE FROM [ ONLY ] table [ WHERE condition ]
```

expression for set difference operand; corresponds to a relational select on *table*

```
INSERT INTO table [ ( column [, ...] ) ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) | query }
```

expression for union: literal tuples, database query (see SQL *select* statement)

```
UPDATE [ ONLY ] table SET column = { expression | DEFAULT } [, ...]
    [ FROM fromlist ]
    [ WHERE condition ]
```

- generalized projection
- relational select condition
- implicit union of unaffected tuples

Data Manipulation: Queries

In the single *select* statement, SQL combines the entire range of relational operators: projection (generalized), selection, union, set-difference, cartesian-product, rename, set-intersection, natural-join (theta), outer join (theta), aggregate functions — plus some practical extras

- *select* clause: generalized projection
- *from* clause: relations, cartesian-product, natural-join, outer join
- *where* clause: selection (predicate)
- *group by* clause: aggregate functions
- *having* clause: selection on aggregate function results
- *union*, *intersect*, *except*: relational union, intersection, and set-difference, respectively

– no SQL equivalent for relational division

\+ *order by*: sorting of tuple results (relational theory doesn't care, since relations are sets)

\+ *distinct*, *limit*: modifies which tuples are returned (*distinct* = no duplicates; *limit* = maximum)

generalized projection, aggregate functions

predicate for relational select

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ AS output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start ]
    [ FOR UPDATE [ OF table_name [, ...] ] ]

where from_item can be one of:

    [ ONLY ] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    ( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
    function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias
[, ...] | column_definition [, ...] ) ]
    function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
    from_item [ NATURAL ] join_type from_item [ ON join_condition | USING
( join_column [, ...] ) ]
```

aggregate functions: grouping, select

set-like operators

relational rename (within the query)

joins: natural, outer, theta

return unique tuples only

formally, the *where* clause performs a relational select on the cartesian-product of the *from_items*

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ AS output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start ]
    [ FOR UPDATE [ OF table_name [, ...] ] ]
```

specific sort order for returned tuples

return a maximum number of tuples

nested query: composition of relational operations (a.k.a. intermediate results)

choose between *inner join*, *left/right/full outer join*

```
where from_item can be one of:

    [ ONLY ] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    ( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
    function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias
[, ...] | column_definition [, ...] ) ]
    function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
    from_item [ NATURAL ] join_type from_item [ ON join_condition | USING
( join_column [, ...] ) ]
```

formal natural-join = *natural inner join*

theta join (θ = *join_condition*)

equality on any set of attributes

Set Comparisons, Nested Queries

- *in (nested_query)*: tests for inclusion in the result of the nested query
- >, <, >=, <=, =, <> *some*, *all (nested_query)*: compares an expression across the tuples in the nested query
  - Use in the *where* clause to compare expressions based on attributes in *from* clause relations
  - Use in the *having* clause to compare aggregate function results
- *exists*, *not exists*: tests for empty/non-empty relations

Odds & Ends

- String comparisons: *like* comparator allows use of wildcards ('%' = any substring, '_' = any character)
- String functions: || performs concatenation; other conventional-looking functions like *upper()* and *lower()* are also available
- Aggregate functions: *avg*, *min*, *max*, *sum*, and *count* are built-in; use *count(*)* to count tuples
- Boolean operators: *and*, *or*, and *not* work as expected
- Null values: *is null* and *is not null* perform null tests