Relational Algebra

Unary: Selection σ_c | Projection π_ℓ | Renaming ρ_ℓ
Binary: Union $\cup$ | Intersect $\cap$ | Difference $-$ | Cross Product $\times$
Join: Inner $\bowtie_c$ | Natural $\bowtie$ | Left/Right/Full $\rightarrow_c \leftarrow_c \leftrightarrow_c$ | Natural Left/Right/Full $\rightarrow \leftarrow \leftrightarrow$
$\textbf{null}$: Comparison operation: $\textbf{unknown}$ Arithmetic operation: $\textbf{null}$

ER Model

Relationship constraints:
Key (arrow): Many-To-Many | One-To-Many/Many-To-One | One-To-One
Participation (thickness): partial thin line | total thick line
Weak entity sets:
No own key, dependent on owner entity, relationship must have total participation
Aggregation: relationship between entities and relationships
ISA hierachy:
Overlap constraint: can belong to multiple subclasses? Undirected if yes, directed if no
Covering constraint: must belong to some subclass? Thick if yes, thin if no

SQL

Transactions ACID:
Atomicity: either all effects are reflected or none are
Consistency: preserved by execution in isolation
Isolation: isolated from effects of concurrent transactions
Durability: effects persists even during system failures
Constraint types: Not null, Unique, Primary key, Foreign key, Check
Subqueries: `EXISTS` `IN` `ANY/SOME` `ALL`
Used in `WHERE`, `FROM` (enclosed with parentheses and given alias), `HAVING`
Aggregate functions:
Used in `SELECT`, `HAVING`, `ORDER BY` or subqueries
`COUNT(A)`: number of non-null values `COUNT(*)`: number of rows
`GROUP BY` clause: For each column A in `SELECT` or `HAVING`
A in `GROUP BY`, or A in an aggregated expression, or primary key in `GROUP BY`
Pattern matching: `_`: single character `%`: 0 or more characters
Evaluation of queries:

```
SELECT select-list FROM from-list WHERE where-condition GROUP BY group-by-list
HAVING having-condition ORDER BY order-by-list LIMIT ... OFFSET ...
```

I. Compute cross-product in `from-list`
II. Select tuples for `where-condition`
III. Partition using `group-by-list`
IV. Select groups with `having-condition`
V. Generate output for attributes in `select-list`
VI. Remove duplicate tuples

VII. Sort output tuples based on `order-by-list`
VIII. Remove tuples based on `OFFSET` and `LIMIT`

```
x IS DISTINCT FROM y -- false if both null, true if one null
UNION / INTERSECT / EXCEPT -- eliminate duplicates
UNION ALL / INTERSECT ALL / EXCEPT ALL -- INTERSECT higher precedence
CASE [expression]
  WHEN condition_i/value_i THEN result_i WHEN ... THEN ... ELSE result_j
END;
NULLIF(value_1, value_2) -- NULL if value_1 = value_2, otherwise value_1
```

PL/pgSQL

Benefits: code reuse, ease of maintenance, performance, security

```
/* functions */
CREATE OR REPLACE FUNCTION <f_name> (<param> <type>, ...) RETURNS <type> AS $$
... $$ LANGUAGE sql;
SELECT <f_name>(...); /* tuple */ SELECT * FROM <f_name>(...); /* table */
/* variants */
RETURNS SETOF <type> /* more than one tuple */
(IN <param> <type>, OUT <output> <type>, ...) /* custom tuple */
RETURNS TABLE(<param> <type>, ...) /* simplified */
/* procedures */
CREATE OR REPLACE PROCEDURE <p_name> (<param> <type>, ...) AS $$
... $$ LANGUAGE sql;
CALL <p_name>();
```

```
DECLARE <name> <type> := <value>;
BEGIN
  SELECT ... INTO <val> FROM ...;
  <name> := ...;
END;
RETURN QUERY SELECT ...; /* return set of tuples, does not exit */
RETURN NEXT; /* return set of tuples, does not exit */
```

```
/* cursor */
DECLARE curs CURSOR FOR (SELECT ... FROM ...);  r RECORD;
BEGIN OPEN curs;  ...  CLOSE curs; END;
FETCH curs INTO r; FETCH NEXT FROM curs INTO r;
FETCH PRIOR/FIRST/LAST FROM curs INTO r;
FETCH ABSOLUTE/RELATIVE <num> FROM curs INTO r;
```

Triggers

```
CREATE OR REPLACE FUNCTION trigger_func() RETURNS TRIGGER AS $$
DECLARE ...
BEGIN
  -- NEW: new tuple being inserted/updated
  -- OLD: old tuple being deleted/updated, null for insert
  -- TG_OP: INSERT|UPDATE|DELETE  TG_TABLE_NAME: table that invoked
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_name
[BEFORE|AFTER|INSTEAD OF] INSERT|UPDATE|DELETE [OR ...] ON table_name
FOR EACH ROW|STATEMENT EXECUTE FUNCTION trigger_func();
```

Trigger timing:

`BEFORE`: non-null: normal operation   null: operation ceased

`AFTER`: return value does not matter

`INSTEAD OF`: non-null: normal operation   null: ignore rest of the operation

Trigger levels:

Row-level: `INSTEAD OF` not allowed

Statement-level: use `RAISE EXCEPTION` to omit subsequent operations

Trigger condition:

```
CREATE TRIGGER trigger_name BEFORE ... FOR EACH ROW
WHEN (condition) EXECUTE FUNCTION trigger_func();
```

No `SELECT` | No `OLD` for `INSERT` | No `NEW` for `DELETE` | No `WHEN` for `INSTEAD OF`

Trigger order:

`BEFORE` statement, `BEFORE` row, `AFTER` row, `AFTER` statement

Activated alphabetically within each category

If `BEFORE` row returns `NULL`, subsequent triggers on the same row omittedRequirements:

Deferred trigger:

Only works with `AFTER` and `FOR EACH ROW`

```
CREATE CONSTRAINT TRIGGER trigger_name AFTER ...
DEFERRABLE INITIALLY DEFERRED|IMMEDIATE
FOR EACH ROW EXECUTE FUNCTION trigger_func();
```

FD

Armstrong's Axioms:

Reflexivity: $AB \to A, AB \to B$

Augmentation: $A \to B \Rightarrow AC \to BC$

Transitivity: $A \to B, B \to C \Rightarrow A \to C$

*Decomposition: $A \to BC \Rightarrow A \to B, A \to C$

*Union: $A \to B, A \to C \Rightarrow A \to BC$

Closure: To prove $X \to Y$, simply show $Y \in \{X\}^+$

Finding keys: Enumerate all subsets, Compute closures, Identify superkeys, Identify keys

Check small attribute sets first

If A does not appear in the RHS of any FDs, A must be in key

BCNF

Definition: every NT&D FD has a superkey as its LHS

Checking: "more but not all"

I. Compute the closure of each attribute subset

II. Check if exists $\{A_1A_2 \ldots A_k\}^+$ that satisfies "more but not all"

III. If such closure exists, R is not in BCNF

Decomposition algo:

I. Find an attribute subset X such that $\{X\}^+$ satisfies "more but not all"

II. Decompose into two tables R_1 and R_2

R_1 contains all in $\{X\}^+$

R_2 contains all in X and attributes not in $\{X\}^+$

III. If R_1/R_2 not in BCNF, further decompose R_1/R_2

To derive the FDs on R_1/R_2, project closures of R on them

2-attribute tables must be in BCNF, no need to check

Lossless join decomposition: common attributes in R_1 & R_2 constitute a superkey of R_1/R_2

Minimal Basis

Definition:

I. Every FD in M can be derived from S and vice versa

II. Every FD in M is NT&D

III. If any FD removed from M, some FD in S cannot be derived

IV. For any FD in M, if attribute removed from LHS, FD cannot be derived from S

Algo:

I. Decompose all FDs so that RHS contains only one attribute

II. Remove redundant attributes on LHS (hide attribute A, redundant if FD implied by S)

III. Remove redundant FDs (hide FD, redundant if FD implied by existing FDs)

3NF

Definition: every NT&D FD **either** LHS is superkey **or** RHS is prime attribute

Checking:

I. Compute closure for each attribute subset

II. Derive of keys of R

III. For each closure $\{X_1, \ldots, X_k\}^+ = \{Y_1, \ldots, Y_m\}$, check if

$\{Y_1, \ldots, Y_m\}$ does not contain all attributes, and

Exists an non-prime attribute in $\{Y_1, \ldots, Y_m\}$ that is not in $\{X_1, \ldots, X_k\}$

IV. If such closure does not exist, R is in 3NF

Decomposition algo:

I. Derive minimal basis of S

II. Combine FDs whose LHS are the same

III. Create a table for each FD

IV. If none of the tables contains a key, create a table that contains any key

V. Remove any redundant table