SQL Queries (6.1-6.5)

Trick: from, the where, the select.
SELECT and FROM are required.
FROM: clause gives the relation or relations to which the query refers.
WHERE: clause is a condition, much like a selection-condition in relational algebra.
SELECT: clause tells which attr. of the tuples matching the cond. are produced as part of the answer. (proj.)

Projection In SQL

SELECT * FROM R; A1, A2
Alias: SELECT A2 AS B2 FROM R;
SELECT list can function like th list in an extended projection and can also include aggregation.
SELECT length*0.15 AS Time FROM R;

Can add constant as an expression in SELECT clause Ex. 'hrs.' AS inHours.
Adds a column with hrs. in every tuple.
SQL is case insensitive, unless in quotes.
Order of Precedence: NOT, AND, OR

Selection In SQL

Operators: =, <>, <, >, <=, and >=.
- <> SQL symbol "not equal to".
- = SQL Symbol ==.
- || Concatenation operator to string.

Arithmetic operators are allowed.

SELECT L FROM R WHERE C;
→ $\pi_L(\sigma_C(R))$

= for strings is the same sequence of CHAR.
<, >= precedes in lexicographic order.
'bar' < 'bargain', 'fodder' < 'foo'.

Pattern Matching: S NOT LIKE P, where s = string and p = pattern.
- % 0 or more
- _ any 1 CHAR
- '' is a single apostrophe
- S LIKE 'x' % :% x % ESCAPE 'x'

→ % is in th matching string.

DATE '1995-05-14'
TIME '15:00:02.5' or '12:00:00-8:00'
TIMESTAMP combines the above.

NULL: Value unknown, value inapplicable, value withheld.
- operator with NULL results in NULL.
- Compare with NULL results in UNKNOWN.

Use X IS NULL, X IS NOT NULL
△ TRUE = 1, FALSE = 0, NULL = 0.5
Thus, AND is min, OR is max, NOT depends on X.
— Only tuples with condition with TRUE are included.

x	y	x AND y	x OR y	NOT x
T	T	T	T	F
T	U	U	T	F
T	F	F	T	F
U	T	U	T	U
U	U	U	U	U
U	F	F	U	U
F	T	F	T	T
F	U	F	U	T
F	F	F	F	T

ORDER BY: τ operation, < list of attr. >
- ASC, DESC

ORDER BY clause follows WHERE clause and any other clause (GROUP BY, HAVING).
△ SELECT applied after ORDER BY.
ORDER BY A+B DESC with R(A, B)

SQL With More Than One Relation

SELECT * FROM A, B WHERE C;
Can use R.A for disambiguating attr. in SELECT.
Tuple Variable: FROM R R1, R AS R2
Nested loops, parallel assignment.
Note: <> produces pairs twice.
SELECT R.A FROM R,S,T WHERE R.A = S.A OR R.A = T.A;
Note: If T is empty, nothing is returned.
- UNION ∪, INTERSECT ∩, EXCEPT −

(Query) UNION / INTERSECT / EXCEPT (Query)

Subqueries

Subqueries can have subqueries. They return single constant, WHERE returns relations, WHERE in FROM followed by tuple var.
Subqueries produce scalar values. If 0 or <2 is returned the run-time errors.

EXISTS R: is a condition that is true iff R is not empty. (NOT can be applied to below).
△ S IN R: true iff S is equal of the value in R.
△ S NOT IN R: true iff s = equal to no value in R.
△ S > ALL R: iff S > every column in R.
△ S > ANY R: iff s > any value in R.
△ T IN R, t <> ANY R, for tuples.

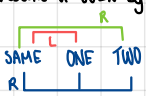
Correlated Subqueries

Subquery to be evaluated many times.
Scoping rule for names.
△ OLD belong to another tuple variable.
- SUBQUERIES in FROM clauses.
- Relations in SQL are bags, not sets.

SQL JOIN Expressions

- products, natural joins, theta join, outer join.

△ CROSS JOIN: R with all attributes
△ Theta Join: JOIN . . ON ... is R X S.
→ duplica values; R1 JOIN R2 ON C.
NATURAL JOIN: R1 ⋈ R2 All pairs of attributes have common name are equated, no other condition, each pair of equated attribute is projected out. (Inherent / WHERE ON)
OUTER JOIN: Augment results of JOIN by dangling tuples, i.e., NULL.



All Tuples:
HAVING: clause allows selection of certain groups in a whole group. 6 operator.
DISTINCT: SELECT DISTINCT A to eliminate duplicates. U, ∩, − usually eliminate duplicates. Use ALL to avoid behaviour. Note: It is expensive to eliminate duplicates.
Aggregation: SUM, AVG, MIN, MAX, COUNT
- COUNT(*) counts all the tuples.
- COUNT(DISTINCT x)

SELECT X FROM Y GROUP BY X;
- In a SELECT clause that has aggregation only those attributes that are mentioned in th GROUP BY clause may appear unaggregated in th SELECT clause.

NULL ignored in aggr. (FOR NON-NULL COUNT(*))
NULL is ordinary in GROUP BY. Empty bag is 0.
SUM(NULL) = NULL.
SELECT, FROM, WHERE, GROUP BY, HAVING, and then ORDER BY.
WHERE C HAVING E: Any tuple that doesn't condition is not. Only tuples being tested. Attributes that are in th GROUP BY list may appear unaggregated in th HAVING. FROM may be aggregated in th HAVING.

Modification

INSERT INTO R(A) VALUES (v);
Default values for attributes not in query. The order matters. Use SELECT to add multiple tuples (no duplicates usually).
DELETE FROM R WHERE C;
UPDATE R SET <n v exp.> WHERE C;

Transactions (6.6)

START TRANSACTION ... COMMIT (ROLLBACK)
Serializability: Locking serializable almost impossible
Atomicity: All or Nothing
SET T READ ONLY; (READ WRITE)
Dirty Reads: Not committed (READ ONLY default)
SET TRANSACTION LEVEL READ COMMITTED / REPEATABLE READ, SERIALIZABLE, READ UNCOMMITTED

Constraint and Triggers (7)

FK must be declared UNIQUE or PK. FK must appear in reference attr. of some tuple.
REFERENCES <table> (<attrs.>)
FOREIGN KEY (<attrs.>)

Referential Integrity

INSERT R1 WHERE FK IS NOT IN R2
UPDATE R1 WHERE FK IS NOT IN R2
UPDATE R2 WHERE FK IS IN R1
DELETE R1 WHERE FK IS IN R2

Constraints

DEFAULT policy: reject violating mods.
CASCADE policy: changes to the ref attrs. are mimicked at th FK.
SET NULL policy: mod. to referred R affect a FK value, changed to NULL. CREATE ON D SET NULL.
INSERT with NULL bypass constraints.
DEFERRABLE initially def or immediate.
SET CONSTRAINT myConstraint DEFFERED;
CREATE TABLE NOT NULL A CHECK(A);
CREATE TABLE ... CHECK(condition);
CONSTRAINT myConstraint PRIMARY KEY;
We cannot add a constraint to a table unless it holds.
ALTER TABLE R DROP CONSTRAINT myConstraint.
ALTER TABLE R ADD CONSTRAINT myConstraint Const,
CREATE ASSERTION <ass.> CHECK(cond.);

Triggers (event condition action)

CREATE TRIGGER AFTER, BEFORE → UPDATE
INSTEAD OF, FOR EACH ROW, EACH STATEMENT OF.
WHEN clause optional.
BEGIN ... END
Row-Level: update (old, new), insert (new), deletion (old).
Statement-Level: No old or new.
OLD TABLE AS OLD, NEW T AS NEW.

Views and Indexes (8.1-8.4)

Virtual views are not stored. Materialized views = index.
CREATE VIEW <v.n> AS <v.def>
CREATE VIEW <v.n> (attrs.) AS ...
INSERT, DELETE, and UPDATE

Updatable Views: Mod view → Mod base.
WHERE clause not involve R in subquery.
FROM clause one occurrence of R and no other.
Note: can't project NOT NULL or NO DEFAULT.
Removal: DROP VIEW v_n;
DROP TABLE, means view becomes unusable.
TRIGGER on view use INSTEAD OF for
BEFORE and AFTER.

Indexes: Efficient to find tuples. Binary Search
Tree of (Key, Value). (multi-attr. indexes).

CREATE INDEX name ON TABLE <attrs.>;
DROP INDEX name;

Useful: queries value is common one or zero.
Clustered: grouping th tuples.

Materialized Views: Maintain values.
CREATE MATERIALIZED VIEW name;
Changes are incremental FROM clause.

ACID Properties

Atomicity: Be indivisible.
Consistency: Not leave DB in inconsistent state.
Isolation: Independant.
Durability: Permanent after commit

Midterm 03 - SQL & Transactions

WHERE AFTER ON BEFORE JOIN
SHOW CREATE TABLE;
DELIMITER $$;
DESCRIBE TABLE;
DEFAULT
ALTER TABLE R DROP A;
ALTER TABLE R ADD A TYPE
ALTER TABLE R MODIFY type A(...)
DELETE FROM R;
Think about PK being auto-incremented.
INNER JOIN usually throws away NULL.
NATURAL JOIN throws away duplicate attrs.
No new in AFTER TRIGGER.
INSERT only affects tuples inserted.
Delete View also Delete Table.
ADD INDEX is invalid!
Unique, Foreign Key auto constructs an index.