================ Relational Algebra ================

THETA JOIN:
$R_1 \bowtie_C R_2 = \sigma_C(R_1 \times R_2)$

UNION, DIFFERENCE, INTERSECT Operators:
- Schemas must be the same
- No duplicates remain

INTERSECT:
$R \cap S = R - (R - S)$

DIVISION:
$R/S = \pi_A(R) - \pi_A((\pi_A(R) \times S) - R)$
R/S is the largest relation T such that $T \times S \subseteq R$

===================== SQL =====================

General statement:
SELECT attributes, aggregates
FROM relations(tables)
WHERE conditions
GROUP BY attributes
HAVING conditions on aggregates
ORDER BY attributes, aggregates

Evaluation order:
FROM → WHERE → GROUP BY → HAVING → ORDER BY →SELECT

Set Operators:
INTERSECT, UNION, EXCEPT
- Follow set semantics and remove duplicates
- To keep duplicates: UNION ALL, INSERSECT ALL, EXCEPT ALL

Set membership
- IN, NOT IN

Set comparison operator
- ALL, < SOME, = SOME,... etc.

Subqueries
- in WHERE clause
  - Considered as a regular relation
  - One-attribute one-tuple relation -> use like a 'value'
- In FROM clause
  - Considered as a regular relation
  - Must be renamed to a new table name

Aggregates
- Sum, Count, Avg, Min, Max, …

INSERT, DELETE, UPDATE
- Insertion: INSERT INTO <Relation> <Tuples>
- Deletion: DELETE FROM <R> WHERE <Condition>
- Update: Update R
  SET $A_1 = V_1$, $A_2 = V_2$, ..., $A_n = V_n$
  WHERE <Condition>

================ Database Integrity ================

Key Constraints
CREATE TABLE <name> (
…,
PRIMARY KEY(dept, cnum, sec),
UNIQUE(dept, cnum, instructor))

Referential Integrity (Foreign Key)
- E.A references S.A
  - E.A: referencing attribute / foreign key
  - S.A: referenced attribute
- CREATE TABLE <name> (
  …,
  sid INTEGER REFERENCES Student(sid),
  FOREIGN KEY (dept, cnum, sec) REFERENCES Class(dept, cnum, sec)
- Referenced attributes must be PRIMARY KEY or UNIQUE
- RI Violation
  - Default: not allowed
    - System rejects the statement
    - Always insert/update S first
  - ON DELETE/UPDATE SET NULL/SET DEFAULT/CASCADE
    - Added on Referencing attributes declaration
    - SET NULL/SET DEFAULT
      - Change E.A value to NULL or default value
    - CASCADE
      - On deletion of S: delete referencing tuples in E
      - On update of S.A: change E.A to the new S.A

Check Constraints
- CRATE TABLE Enroll (
  dept CHAR(2), cnum INT, unit INT,
  title VARCHAR(50),
  CHECK (cnum < 600 AND unit < 10))

Triggers
CREATE Trigger <name>
<event>
<referencing clause>
WHEN (<condition>)
<action>

<event>
- BEFORE | AFTER INSERT/DELETE/UPDATE [OF A1, A2, …, An] ON R

<referencing clause>
- REFERENCING OLD | NEW TABLE | ROW AS <var>, …
- FOR EACH ROW|STATEMENT

<action>
- Any SQL statement

================ Views and Authorization ================

Views
CREATE VIEW <name> AS
<Query>

Authorization
- GRANT <privileges> ON <R> TO <user> [WITH GRANT OPTION]
- REVOKE <privileges> ON <R> FROM <user> [CASCADE | RESTRICT]

================ Misc ================

Logical Implies in SQL
$p \to q \equiv \neg p \vee q$

================ Files and Disks ================

Access time = (seek time) + (rotational delay) + (transfer time)

- Seek time: time to find the target track
  - Typical average seek time: 10 ms
- Rotational delay: time to rotate to the target sector
  - For 6000 RPM, average rotational delay=0.5*(1min/6000)=0.5*60sec/6000=5 ms
- Transfer Time
  - Time to read one block
  - For example, 6000 RPM, 1000 sector/track, 1KB/sector
    - Read a track, rotate a circle: 1min/6000 = 10 ms/track
    - Read one sector(block): (10ms/track) / (1000sector/track) = **0.01ms/sector**
    - Transfer rate: 1KB/(0.01ms/sector)=**100MB/s**

================ B+ Trees ================

Insertion:

- Leaf node overflow
  - The first key of the new node is ***copied*** to the parent
- Non-leaf node overflow
  - The middle key is ***moved*** to the new parent

	MaxPtrs	MaxKeys	MinPtrs	MinKeys
Non-leaf Non-root	n	n-1	⌈n/2⌉	⌈n/2⌉-1
Leaf Non-root	n	n-1	⌈(n+1)/2⌉	⌈(n-1)/2⌉
Root	n	n-1	2	1

Deletion:

- Try to merge first, if not, redistribute
  - Merging is always moving from right to left
- Leaf node merging
  - ***Delete*** the mid-key from the parent
- Non-leaf node merging/redistribution
  - ***Pull down*** the mid-key from the parent first
  - Then move from right to left

Min, max:

Min records with k levels (k ≥ 2): $2 \times \lceil n/2 \rceil^{k-2} \times \lceil (n-1)/2 \rceil$

Max records with k levels (k ≥ 2): $n^{k-1} \times (n-1)$

Ex.

Consider a B+tree that indexes 300 records. Assume that n = 5 for this B+tree (i.e., each node has at most 10 pointers), what is the minimum and maximum height (depth) of the tree? (A tree with only the root node has a height of 1.)

ANSWER:

Minimum 4. (maximum 4 record pointers per node at leaf. Ceil(300/4) = 75 leaf nodes are needed when full. maximum branching factor 5 at non-leaf nodes. Ceil(75/5) = 15 nodes are needed at level 2. Ceil(15/5) = 3 nodes are needed at level 3. One more level of root node that points to these three nodes.)

Maximum 5. (minimum 2 record pointers per node at leaf. Flr(300/2) = 150 leaf nodes. Minimum branching factor 3 at non-leaf nodes. Flr(150/3) = 50 nodes at level 2. Flr(50/3) = 16 nodes at level 3. Flr(16/3) = 5 nodes at level 4. Flr(5/3) = 1 nodes at level 5. Since there is only one node at level 5, this is the root node.)

================ Join ================

Nested-Loop Join

- For each r ∈ R do
  For each s ∈ S do
  if r.C = s.C then output r,s pair
- Use smaller table for outer loop (R)
- Bulk block NLJ
  - $b_R + \lceil b_R/(M-2) \rceil \times b_S$

Hash Join

- Hashing stage (bucketizing)
  - Hash R tuples into G1,...,Gk buckets
  - Hash S tuples into H1,...,Hk buckets
- Join stage
  - For i = 1 to k do
  - match tuples in Gi, Hi buckets
- Number of buckets = M-1
- General cost ($b_R < b_S$)
  - $2(b_R + b_S)\left\lceil \log_{M-1}\left(\frac{b_R}{M-2}\right)\right\rceil + (b_R + b_S)$

Index Join

- Cost:
  - IO for R scanning
  - IO for index look up
  - IO for tuple read from S
- General cost:
  - $b_R + |R| \times (C + J)$
  - C average index lookup cost
  - J matching tuples in S for every R tuple

Sort-Merge Join

1. Sort stage: Sort R and S
2. Merge stage: Merge sorted R and S

- General cost:

$$2b_R\left(\left\lceil \log_{M-1}\left(\frac{b_R}{M}\right)\right\rceil + 1\right) + 2b_S\left(\left\lceil \log_{M-1}\left(\frac{b_S}{M}\right)\right\rceil + 1\right) + (b_R + b_S)$$

In general:

- Nested-loop join ok for “small" relations (relative to memory size)
- Hash join usually best for equi-join
  - if relations not sorted and no index
- Merge join for sorted relations
  - Sort merge join good for non-equi-join
- Consider index join if index exists

================ Transaction ================

Atomicity: all or nothing.

Consistency: If the database was in consistent state, it remains in consistent state.

Isolation: The end result is the same as when transactions are run in isolation.

Durability: Results from committed transactions are never lost.

================ E/R Model ================

N-ary Relationships

- Arrow in a a N-ary relationship: pick one entity from every other set without arrow. Together, these entities must be related to at most one entity with arrow.
- Do not put multiple arrows for non-binary relationships. Very confusing. No standard interpretation.

Subclasses

- Subclass inherits all attributes of its superclass
- Subclass participates in the relationships of its superclass
- Subclass may participate in its own relationship
- Total Specialization: Double lines in E/R. Entity is always one of subclasses

Weak Entity Set

- Entity sets without unique keys
    - o Notation: Double rectangle and double diamond in E/R
    - o A part of its key comes from one or more entity set it is linked to
- Discriminator: a set of attributes in W.E.S. that are part of the key
    - o Dashed underline in E/R
- Owner Entity Set: entity set providing a part of the key
- Identifying Relationship: relationship between a weak entity set and owner entity set
    - o Always double edge between a weak entity and identifying relationship

E/R to Relation

- (STRONG) ENTITY SET: one table with all attributes
- RELATIONSHIP SET: one table with keys from the linked ES and its own attributes
    - o Rename attributes when names conflict, like TA.name and Student.name
    - o Use role label as attribute names
- WEAK ENTITY SET: one table with its own attributes and keys from owner ES
    - o No need to translate identifying relationship set
- SUBCLASS: three approaches
    1. one table for each subclass with all its attributes plus key from its superclass (Student, ForeignStudent, HonorStudent)
    2. one big relation with all attributes with null values for missing attributes (Student)
    3. one table for every subtree (including the root) with all its attributes plus all "inherited" attributes (Student, FStudent, HStudent, FHStudent)

================ Normalization Theory ================

Functional Dependency

- For any u1, u2 ∈ R, if u1[X] = u2[X], then u1[Y] = u2[Y]
- TRIVIAL functional dependency: X → Y when Y ⊂ X
- NON-TRIVIAL FD: X → Y when Y !⊂ X
- COMPLETELY NON-TRIVIAL FD: X → Y with no overlap between X and Y
- X is a KEY of R if and only if
    1. X → all attributes of R (i.e., X+ = R)
    2. No subset of X satisfies 1 (i.e., X is minimal)
- PROJECTING FD
- In order to find FD's after projection, we first need to compute F + and pick the FDs from F + with only the attributes in the projection.

Decomposition

- DECOMPOSITION R(X, Y, Z) ⇒ R1(X, Y), R2(X, Z) IS LOSSLESS IF X → Y OR X → Z
    - o That is, the shared attributes are the key of one of the decomposed tables
    - o We can use FDs to check whether a decomposition is lossless
        - ▪ When checking, check FDs on original table

BCNF

- R is in BCNF with regard to F, iff for every non-trivial X → Y , X contains a key
- Normalization
    - o Decomposing tables until all tables are in BCNF
        - ▪ For each FD X → Y that violates the condition, separate those attributes into another table to remove redundancy
        - ▪ We also have to make sure that this decomposition is lossless
    - o Algorithm
        - ▪ For any R in the schema
        - ▪ If non-trivial X → Y holds on R, and if X does not have a key
            1. Compute X+ (X+: closure of X)
            2. Decompose R into R1(X+) and R2(X, Z) // X is common attributes where Z is all attributes in R except X+
        - ▪ Repeat until no more decomposition
    - o NOTE: We have to check all implied FD's for BCNF, not just the given ones

MVD X->>Y

- Definition: for every tuple u, v ∈ R:
    - o If u[X] = v[X], then there exists a tuple w such that:
        1. w[X] = u[X] = v[X]
        2. w[Y] = u[Y]
        3. w[Z] = v[Z] where Z is all attributes in R except X and Y
    - o X->>Y means that if two tuples in R agree on X, we can swap Y values of the tuples and the two new tuples should still exist in R.
- COMPLEMENTATION RULE: X->>Y , then X->>Z where Z is all attributes in R except X and Y
- TRIVIAL MVD: X->>Y is trivial MVD if
    1. Y ⊂ X -or-
    2. X ∪ Y = R

4NF

- Definition: R is in 4NF if for every nontrivial FD X → Y or MVD X ->> Y , X contains a key
- First, using all functional dependencies, normalize tables into BCNF. Then apply the following algorithm to normalize them further into 4NF.
  For any R in the schema
    If non-trivial X ->> Y holds on R, and if X does not contain a key
    Decompose R into R1(X, Y) and R2(X, Z) // X is common attributes where Z is all attributes in R except (X, Y)
  Repeat until no more decomposition