Database Fundamentals & Relational Model

Data Structured, unstructed, semi-structured
Key Constraints

- **Superkey (SK)**: A set of attributes that uniquely identifies a tuple in a relation.
- **Candidate Key (CK)**: A minimal superkey.
- **Primary Key (PK)**: A candidate key that is chosen to uniquely identify tuples in a relation; cannot be null.
- **Foreign Key (FK)**: A set of attributes that refers to the primary key of another relation.

Entity Integrity Constraint: Ensures that the primary key is not null.
Referential Integrity Constraint: Ensures that the foreign key references a valid primary key in another relation
Operations on Relations
When an operation violates integrity constraints, the system can:

- **Reject/Abort**: Cancel the violating operation entirely, Maintain database consistency
- **Notify Only**: Complete the operation, Alert user about the violation
- **Automatic Correction**: CASCADE: Propagate changes to maintain referential integrity, SET NULL: Set dependent values to null
- **Custom Handler**: Execute user-defined error correction procedures
- **Silent Failure** (Not Recommended): Ignore the violation, Can lead to data inconsistency

Database Constraint Violations

- **INSERT Violations**: Domain: Values don't match defined data types; Key: Duplicate primary key values; Referential: Foreign key references missing primary key; Entity: NULL values in primary key.
- **DELETE Violations**: Referential integrity when deleting referenced records; Options: Restrict: Block deletion; Propagate: Delete corresponding records; Set Null: Nullify foreign keys; Set Default: Use default values.
- **UPDATE Violations**: Primary Key: Creating duplicate keys; Foreign Key: Setting invalid references; Resolution: Validate constraints first; Choose appropriate action (RESTRICT, CASCADE, etc.); Consider business impact.

Guidelines for a Good Design

- The attributes of a relation should make senese
- Avoid redundant tuples
- Relations should have as a few NULL values as possible
- Design relations to avoid fictitious tuples after join

Functional Dependency is a formal metric of the degree of goodness of a relation schema. $X \rightarrow Y$ (X uniquely determines Y) holds if whenever two tuples have the same value for X, they must have the same value for Y If K is a Candiate Key, then $K \rightarrow R$ (A) for all attributes A in R
Partial Dependency: A non-key attribute A is partially dependent on the primary key K if A is dependent on a proper subset of K.
Transitive Dependency: A non-key attribute A is transitively dependent on the primary key K if A is dependent on another non-key attribute B, and B is in turn dependent on K.
Normalization: progressive decomposition of unsatisfactory (bad) relations by breaking up their attributes into smaller good relations. A prime attribute is an attribute that is part of some candidate key.
First Normal Form (1NF): A relation is in 1NF if the domain of each attribute is atomic (cannot be decomposed) and each tuple is unique. This does not allow nested or multi-valued attributes.
Second Normal Form (2NF): A relation is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key.
Third Normal Form (3NF): A relation is in 3NF if it is in 2NF and there are no transitive dependencies.
Boyce-Codd Normal Form (BCNF): A relation is in BCNF if it is in 3NF and all determinants are primary keys.
SQL
Database Schema and Table: 'CREATE SCHEMA' defines a namespace for database objects; 'CREATE TABLE' defines the structure of a table. **Data Types**: 'INTEGER', 'FLOAT', 'CHAR', 'VARCHAR', 'BOOLEAN', 'DATE', 'TIME', 'TIMESTAMP' **Constraints**: 'PRIMARY KEY', 'FOREIGN KEY', 'UNIQUE', 'CHECK (condition)', 'NOT NULL', 'DEFAULT value' **Multi-set operators**: 'UNION', 'INTERSECT', 'EXCEPT'
Any value compared with NULL is unknown, should use 'IS NULL' or 'IS NOT NULL' to check for NULL values.
Joins

- **Inner Join**: Returns only the tuples that match the join condition. 'WHERE table1 INNER JOIN table2'
- **Left (Outer) Join**: Returns all tuples from the left relation and the matching tuples from the right relation. 'WHERE table1 LEFT OUTER JOIN table2 (ON condition)'
- **Right (Outer) Join**: Returns all tuples from the right relation and the matching tuples from the left relation. 'WHERE table1 RIGHT OUTER JOIN table2 (ON condition)'
- **Full (Outer) Join**: Returns all tuples from both relations. 'WHERE table1 FULL OUTER JOIN table2'

Nested Queries: A query that is nested inside another query.

- **In**: Returns true if the value is in the subquery. 'WHERE column IN (SELECT column FROM table2)'
- **Exists**: Returns true if the subquery returns at least one tuple. 'WHERE EXISTS (SELECT * FROM table2 WHERE column = value)'

Aggregate Functions: 'COUNT', 'SUM', 'AVG', 'MAX', 'MIN'
Grouping: 'GROUP BY'
HAVING: Used to filter groups based on aggregate functions.
Window Functions: `ROW_NUMBER ()`, `RANK ()`, `DENSE_RANK ()`, `LEAD ()`, `LAG ()`. **Window Functions**: Window functions perform calculations across a set of rows related to the current row, providing a value for each row instead of summarizing data.

```
function_name() OVER (PARTITION BY column1 ORDER BY column2)
```

- **Function**: The calculation to perform (e.g., `ROW_NUMBER ()` or `SUM ()`).
- **OVER**: Indicates the use of a window function.
- **PARTITION BY**: Groups the data for the function. If omitted, the entire dataset is treated as one group.
- **ORDER BY**: Determines the order of rows within each group.
- **Window Frame**: Defines the rows to operate on. (ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) (BETWEEN 1 PRECEDING AND 1 FOLLOWING)

Physical Design and Hashing
Organisation based Optimisation. Records are grouped together formating a Block, a file is a group of blocks. blocking factory = $\lfloor |B|/|R| \rfloor$. Number of blocks required = $\lceil |numtuples|/blockingfactor \rceil$. **Linked Allocation**: Each block has a pointer to the next block.
File Structures
Heap File: Blocks are stored in an arbitrary order.
Ordered File: Blocks are stored in a specific order. Inserting O (log n) + (move all the blocks), Retrieving (ordering field) O (log n), Retrieving (non-ordering field) O (n). Deleting (ordering field) O (n), non-ordering field O (log n). Can use chain pointers to link records in the same block (sorted linked list)
Hash File: Blocks are stored in a hash table. Inserting O (1), Retrieving O (n), Deleting O (n). Can also use chain pointers to link records in the same block (sorted linked list)
Expectation of Random Variable (Used for Hashing) = $\sum_{i=1}^{n} p_i x_i$
Indexing Methodology
Dense Index: An index entry for every record **Sparse Index**: An index entry for some records
Index Types

- **Primary Index**: index field is ordering, key field of a sequential file. Anchor records: Sparse index, one per block.
- **Clustering Index**: index field is ordering, non-key field of a sequential file. One index per distinct clustering value. Block pointer points at the first block of the cluster. The other blocks of the same cluster are contiguous and accesed via chain pointers.
- **Secondary Index**: index field is:
  - non-ordering, key field, over an ordered or a non-ordered file.
  - non-ordering, non-key field, over an ordered or a non-ordered file.

Multilevel Index: We can build a primary index over any index file.
Multilevel Index: Can become unbalanced
B-Tree: Index on non-ordering key: B-Tree node order p splits the searching space up to p subspaces
Node Definition: Node := $\{P_1, (K_1, Q_1), P_2, (K_2, Q_2), \ldots, P_{p-1}, (K_{p-1}, Q_{p-1}), P_p\}$
B+ Tree: Index on non-ordering key: Internal nodes have no data pointers, only leaf nodes hold data pointers. Has higher fan out. Num pointers is blocking factor.
Internal Node Definition: $p := \{P_1, K_1, P_2, K_2, \ldots, P_{p-1}, K_{p-1}, P_p\}$. Size of internal node is p (pointer size) + $p-1$ (key size)
Leaf Node Definition: $p_L := \{(K_1, Q_1), (K_2, Q_2), \ldots, (K_{p_L}, Q_{p_L}), P_{\text{next}}\}$ Size of leaf node is p (pointer size) + p (key size) + p (sibling pointer size)
When you have many duplicate keys, you should use underground (UG) layer, this means the leaf nodes point to blocks of pointers, which point to the actual data.
External Sorting: Sorting for large relations stored on disk, that cannot fit into memory. Divide and Conquer. Split a file of b blocks into L smaller sub-files. Load each sub-file into memory and sort it. Merge the sorted sub-files into a new sorted file. Cost is $O(2b(1 + \log_M(L)))$. M is degree of merging, L is the number of initial sorted sub-files.
Strategies for Select: Linear search (b/2), binary search (log2 b), primary index (t+1) or hash function (1 + n/2) over a key, hash function over a non key (1 + n (overflow blocks)), primary index over a key in a range query (t + b), clustering index over ordering non-key (t + b/n), secondary index (B+ Tree) over a non-ordering key (t + 1), non-ordering key (t + m + b)
Strategies for Conjunctive Select (AND): if an index exists, use the one that generates the smaller result set, then go through the result set and apply the remaining predicates.
Strategies for Join: Naive join (no index): Compoute the cartesian product, store the results and for each check the join condition nested-loop join (no index): For each tuple in the outer relation, check the inner relation for matching tuples index based nested loop join (index on the inner relation) For each tuple in the outer relation, use the index to find the matching tuples in the inner relation merge-join (sorted relations) Load a pair of sorted blocks, check the join condition and output the result. Efficient if both relations are already sorted on the join key. hash-join (hashed relations) Hash the inner relation and then probe the hash table with the tuples of the outer relation.
Query Optimisation:
Cost-based Optimisation: exploit statistical information to estimate the execution cost of a query. Important is information about each relation and attribute. NDV (Number of Distinct Values).
Selection Selectivity: $0 \leq sl(A) \leq 1$ **Selective Predictions**: Approximation of the selection selectivity. You oculd have no assumption about the data, could be uniformly distributed
Conjunctive Selectivity (A = x and B = y): $sl(Q) = sl(A = x) \cdot sl(B = y) \in [0, 1]$
Disjunctive Selectivity (A = x or B = y): $sl(Q) = sl(A = x) + sl(B = y) - sl(A = x) \cdot sl(B = y) \in [0, 1]$
Selection Selectivity: $\frac{1}{NDV(A)} = \frac{1}{n}$
Selection Cardinality: $\left(\frac{1}{NDV(A)}\right) \cdot r = \frac{r}{n}$
Selection Cost Refinement: Be more accurate: express cost as a function of s(A)
Binary Search on sorted relation: If A is a key, then expected cost is $\log_2(r)$. If A is not a key, then expected cost is $\log_2(b) + \lceil \frac{r \cdot sl(A)}{f} \rceil - 1$.
Multilevel primary index with range A ¿= x: cost: $t + \lceil \frac{r \cdot sl(A)}{f} \rceil - 1$.
Clustering Index over a non key: cost: $t + \lceil \frac{r \cdot sl(A)}{f} \rceil - 1$.
B+ Tree over a no ordering non key: cost: $t + m + r \cdot sl(A)$.
B+ Tree over a no ordering key: cost: $t + 1$.
Multilevel primary index with range A == x: cost: $t + 1$.
Hash file structure: cost: $t + O(n)$.
Join Selectivity Theorem: Given $n = NDV(A, R)$ and $m = NDV(B, S)$: js = $\frac{1}{\max(n,m)}$, jc = $\frac{|R| \cdot |S|}{\max(n,m)}$.