CSE 344 Notes

Contents

1 Introduction

1.1 Databases

A database is a collection of files storing related data

- The rows of a database represent its records
- The columns of a database represent its attributes/fields

1.2 Database Management Systems

A database management system (DBMS) is a program that allows us to efficiently manage a large database and allow it to persist over long periods of time

1.3 Data Models

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another. It consists of the following three parts

- Instance: the actual data
- Schema: a description of what data is being stored
- Query Language: a method of retrieving or manipulating data

1.4 Keys

A key is one or more attribute that, combined, uniquely identify a row

- Primary key: a key that uniquely identifies a row in the current table
- Foreign key: a key that uniquely identifies a row in another table

1.5 Relational Models

A relational model database represents the database as a collection of relations, which is simply a table of values. It consists of the following parts

- Table/Relation
- Columns/Attributes/Fields
- Rows/Tuples/Records

Relational model databases have the following characteristics

- Defined with set semantics such that there are no duplicate tuples
- Attributes are typed and static
- Tables are flat such that there are no subtables

2 Query Languages

2.1 SQL

Structured Query Language (SQL) is a language used to query relational data

- SQL is a declarative programming language

2.2 SQL Queries

- `CREATE TABLE <table> (<attr1> <type1>, ...);`
- `INSERT INTO <table> VALUES (<attr1>, ...), ...;`
- `SELECT <attr1>, ... FROM <table> WHERE <cond>;`
- `DELETE FROM <table> WHERE <cond>;`
- `DROP TABLE <table>;`

2.3 SQL Ordered Queries

`ORDER BY <attr> ASC/DESC` sorts the data based on the order of the attribute

- `SELECT <attr> FROM <table> ORDER BY <attr1> ASC/DESC, ...;`

2.4 SQL Joins

- Inner join: only rows that are shared between both tables are preserved
  - ```
    SELECT <pred1.attr1>, <pred2.attr2>, ...
      FROM <table1> AS <pred1>, <table2> AS <pred2>, ...
      WHERE <cond>;
    ```
- Self join: only rows that occur multiple times in the table are preserved
  - ```
    SELECT <pred1.attr1>, <pred2.attr2>, ...
      FROM <table1> AS <pred1>, <table1> AS <pred2>, ...
      WHERE <cond>;
    ```
- Left outer join: all rows in the left table are preserved
  - ```
    SELECT <pred1.attr1>, <pred2.attr2>, ...
      FROM <table1> AS <pred1>
      LEFT OUTER JOIN <table2> AS <pred2>
      ON <cond> WHERE <cond>;
    ```
- Right outer join: all rows in the right table are preserved
  - ```
    SELECT <pred1.attr1>, <pred2.attr2>, ...
      FROM <table1> AS <pred1>
      RIGHT OUTER JOIN <table2> AS <pred2>
      ON <cond> WHERE <cond>;
    ```
- Full outer join: all rows are preserved
  - ```
    SELECT <pred1.attr1>, <pred2.attr2>, ...
      FROM <table1> AS <pred1>
      FULL OUTER JOIN <table2> AS <pred2>
      ON <cond> WHERE <cond>;
    ```
    ```

2.5 SQL Three-Valued Logic

An SQL predicate can evaluate to true (value 1), unknown (value 0.5), or false (value 0)

- `A AND B` $= \min(\texttt{A}, \texttt{B})$
- `A OR B` $= \max(\texttt{A}, \texttt{B})$
- `NOT A` $= 1 - \texttt{A}$

Comparisons with `NULL` values evaluate out to unknown

- SQL only returns tuples whose condition is true

2.6 SQL Aggregates

`AGG(<attr>)` operates over all non-null values of `attr`

- `SELECT COUNT(*) FROM <table>;`
- `SELECT SUM(<attr>) FROM <table>;`
- `SELECT AVG(<attr>) FROM <table>;`
- `SELECT MIN(<attr>) FROM <table>;`
- `SELECT MAX(<attr>) FROM <table>;`

2.7 SQL Group Aggregates

`GROUP BY <attr>` partitions the data based on matching column values before applying the aggregation

- `SELECT AGG(<attr>) FROM <table> GROUP BY <attr1>, ...;`

2.8 SQL Having Aggregates

`HAVING <cond>` filters groups that do not satisfy the condition

- `SELECT AGG(<attr>) FROM <table> GROUP BY <attr> HAVING <cond>;`
- `SELECT AGG(<attr>) FROM <table> GROUP BY AGG(<attr>) HAVING <cond>;`

`HAVING` filters groups while `WHERE` filters tuples

2.9 FWGHOS

SQL executes queries in the following order

1. `FROM`
2. `WHERE`
3. `GROUP BY`
4. `HAVING`
5. `ORDER BY`
6. `SELECT`

A useful mnemonic to remember this is **F**ish **W**ithout **G**ills **H**ave **O**xygen **S**hortcomings

2.10 SQL Subqueries

SQL subqueries are queries nested in `SELECT`, `FROM`, or `WHERE` which return a table

- `SELECT <pred1.attr1>, ...`
  `FROM <table1> AS <pred1>, (SELECT <pred2.attr2> FROM <table2> AS <pred2>)`
  `WHERE <cond>;`

2.11 SQL Pivots

Pivots convert tables such that values become columns

- `SELECT SUM(CASE WHEN <attr> = <value> THEN 1 ELSE 0 END) AS <label>`
  `FROM <table> WHERE <cond>;`
- `SELECT SUM(<attr> = <value>) AS <label> FROM <table> WHERE <cond>;`

2.12 SQL Set Operators

A set is a collection of distinct items. A bag is a collection of non-distinct items (i.e sets with duplicates)

- `UNION` denotes a set union
- `INTERSECT` denotes a set intersection
- `EXCEPT` denotes a set difference
- `UNION ALL` denotes a bag union
- `INTERSECT ALL` denotes a bag intersection
- `EXCEPT ALL` denotes a bag difference

SQL set operators combine the results of two or more SQL queries

- `(SELECT * FROM T1) <setOp> (SELECT * FROM T2);`

2.13 SQL Set Modifiers

- `ANY` returns true if there exists a tuple in the subquery that satisfies the condition
  - `SELECT ... WHERE value > ANY(subquery);`
- `ALL` returns true if all tuples in the subquery satisfies the condition
  - `SELECT ... WHERE value > ALL(subquery);`
- `IN` returns true if the tuple is in the subquery
  - `SELECT ... WHERE attr IN (subquery);`
- `NOT IN` returns true if the tuple is not in the subquery
  - `SELECT ... WHERE attr NOT IN (subquery);`
- `EXISTS` returns true if the subquery is not equal to the null set
  - `SELECT ... WHERE EXISTS (subquery);`
- `NOT EXISTS` returns true if the subquery is equal to the null set
  - `SELECT ... WHERE NOT EXISTS (subquery);`
- `EXCEPT` returns true for tuples that satisfy the first query but not the second
  - `<query1> EXCEPT <query2>;`

2.14 SQL Matrix Multiplication

Let `A`, `B` be tables with attributes `row`, `col`, `val`

- ```
  SELECT A.row, B.col, SUM(A.val * B.val)
    FROM A, B
    WHERE A.col = B.row
    GROUP BY A.row, B.col;
  ```

2.15 SQL Matrix Addition

Let `A`, `B` be tables with attributes `row`, `col`, `val`

- ```
  SELECT (CASE WHEN A.row is NULL THEN B.row ELSE A.row END) AS row,
     (CASE WHEN A.col is NULL THEN B.col ELSE A.col END) AS col,
     (CASE WHEN A.val is NULL THEN 0 ELSE A.val END) +
     (CASE WHEN B.val is NULL THEN 0 ELSE B.val END) AS val
     FROM A FULL OUTER JOIN B
     ON A.row = B.row AND A.col = B.col;
  ```

2.16 Monotonic Queries

Let I and J be data instances where $I \subseteq J$ and let Q be a query over I. Then Q is a monotonic query if $Q(I) \subseteq Q(J)$

- A monotonic query does not lose/change any tuples with the addition of new tuples in the database
- Aggregate queries are often not monotone

2.17 Monotonicity Theorem

If Q is a `SELECT-FROM-WHERE` query without subqueries and aggregates, then it is monotonic

- If Q is not monotonic, then it is not a `SELECT-FROM-WHERE` query without subqueries and aggregates

2.18 Attribute and Tuple Constraints

Attribute and tuple constraints are data integrity checks that ensure data satisfies a given constraint

- Attribute constraints check that an attribute satisfies a given constraint
  - `CREATE TABLE <table> (<attr1> <type1> CHECK (<cond>), ...);`
- Tuple constraints check that a tuple satisfies a given constraint
  - `CREATE TABLE <table> (<attr1> <type1>, ..., CHECK (<cond>));`

2.19 Referential Constraints

Referential constraints ensure that if a value of one attribute of a relation references a value of another attribute, then the referenced value must exist

- Referential constraints are placed by foreign keys
- `CREATE TABLE <table> (..., <attr1> REFERENCES <attr2> <ref const>);`

2.20 Referential Constraint Maintenance

`ON UPDATE / ON DELETE` maintains references when updating or deleting a row

- Throw an error when updating or deleting a reference row
  - `ON UPDATE / ON DELETE NO ACTION;`
- Update or delete referencers when updating or deleting a reference row
  - `ON UPDATE / ON DELETE CASCADE;`
- Set referencers' field to null when updating or deleting a reference row
  - `ON UPDATE / ON DELETE SET NULL;`
- Set referencers' field to their default values when updating or deleting a reference row
  - `ON UPDATE / ON DELETE SET DEFAULT;`
  - This assumes that a default value was set during the creation of the table

2.21 DDL & DML

- Data Description Language (DDL)
  DDL refers to the commands that set up database tables
  - `CREATE DATABASE ...`
  - `CREATE TABLE ...`
  - `CREATE INDEX ...`
  - `DROP TABLE ...`
  - `ALTER TABLE ...`
- Data Manipulation Language (DML)
  DML refers to the commands that manipulate and query database tables
  - `SELECT ... FROM ...`
  - `INSERT INTO ...`
  - `DELETE FROM ...`

3 Relational Models

3.1 Relational Schema Design

Relational schema design organizes data by consider what data needs to be stored and the interrelationship of the data

3.2 Entity Sets

- An entity is a specific tuple in a data instance
- An entity set is the set of fields that describes an entity
- An attribute is a field in the entity set
- A weak entity uses a foreign key in conjunction with its attributes to create a primary key
- A subclass is an entity set that inherits all the attributes and relationships of its parent entity set

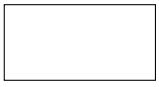
3.3 ER Diagrams

An entity relation diagram (ER diagram) is a model composed of entity types and the relationships that can exist between entities

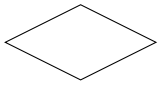
- ER diagrams are rigorous enough that they can be easily converted to SQL

3.4 ER Diagram Building Blocks

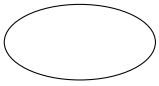
- Entity set



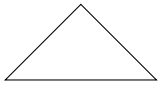
- Relationship



- Attribute

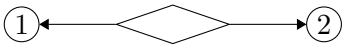


- Subclass

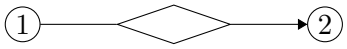


3.5 ER Diagram Relationship Multiplicity

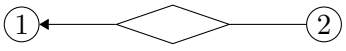
- One-to-one relationship



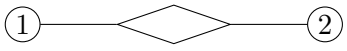
- Many-to-one relationship



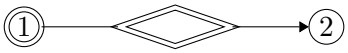
- One-to-many relationship



- Many-to-many relationship



- Weak entity relationship



- One-to-one, one-to-many, and many-to-one relationships do not require relationship tables
- Many-to-many relationships require a separate relationship table with two foreign keys

3.6 Data Anomalies

- Redundancy: redundant rows takes up unnecessary space
  - If rows are redundant, then certain attributes are repeated
- Update: slow updates or inconsistent data
  - If repeated data is present, then multiple updates will be required
- Deletion: deletions are not defined on unrelated columns
  - If data in one attribute is deleted, the whole row might be deleted

3.7 Functional Dependencies

Let R be a relation and $X, Y \subseteq R$ be sets of attributes. Then a functional dependency $X \to Y$ holds in R if the values of Y are uniquely determined by the values of X

- A functional dependency can be thought of as a well-defined function where if $X_1 = X_2$, then $Y_1 = Y_2$
- A functional dependency *holds* on a specific instance of the relation R
- A functional dependency is *satisfied* if it holds on all instances of the relation R

3.8 Armstrong's Axioms

- Axiom of Reflexivity
  - If $B \subseteq A$, then $A \to B$
  - i.e. if $\{\text{name}\} \subseteq \{\text{name}, \text{job}\}$, then $\{\text{name}, \text{job}\} \to \{\text{name}\}$
- Axiom of Augmentation
  - If $A \to B$, then $\forall C,\ AC \to BC$
  - i.e. if $\{\text{ID}\} \to \{\text{name}\}$, then $\{\text{ID}, \text{job}\} \to \{\text{name}, \text{job}\}$
- Axiom of Transitivity
  - If $A \to B$ and $B \to C$, then $A \to C$
  - i.e. if $\{\text{ID}\} \to \{\text{name}\}$ and $\{\text{name}\} \to \{\text{initials}\}$, then $\{\text{ID}\} \to \{\text{initials}\}$

3.9 Closure

The closure of the set of attributes A is the set of attributes B such that $A \to B$

- A closure finds everything a set of attributes determines
- The closure of the set of attributes A is denoted as A^+
- A set of attributes A always determines the attributes contained within itself

3.10 Superkeys

Let R be a relational schema. Then a set of attributes A is a superkey if and only if A^+ is the set of all attributes in R

- A minimal superkey is a set of attributes such that no subset is itself a superkey
  - A key is a minimal superkey

3.11 First Normal Form

A relational schema R is in First Normal Form (1NF) if all attribute values are atomic

- Attribute values cannot be multivalued
- Nested relations are not allowed

3.12 Boyce-Codd Normal Form

A relational schema R is in Boyce-Codd Normal Form (BCNF) if X is a superkey for every non-trivial dependency $X \to A$

- Let C be the set of all attributes in R. Then a relation R is in BCNF if $X^+ = C$ for all X

3.13 Schema Decomposition

Schema decomposition removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables

Let $A_1, ...A_n \to B_1, ..., B_m$. Then the schema can be decomposed as follows

$$R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_k) \begin{cases} \nearrow R_1(A_1, ..., A_n, B_1, ..., B_m) \\ \searrow R_2(A_1, ..., A_n, C_1, ..., C_k) \end{cases}$$

3.14 File Organization

A database consist of a huge amount of data which is stored in physical memory in form of files

- Heap file
  - Heap files are unordered files
  - Tuples are placed in the file in the same order as they are inserted
- Sequential file
  - Sequential files are ordered files
  - Tuples are placed in the file in order, usually by primary key

3.15 Physical Representations

- In a row-store, all the attributes of a single row are stored together. The data may be split into multiple files if the relation is sufficiently large
- In a column-store, all the entries of a single column are stored together. The data always split into multiple files, with each file storing all the entries of a single attribute

3.16 Indexes

An index is a separate file that allows direct access to a row based on attributes

- `CREATE INDEX <index> ON <table>(<attr>);`
- An index is a parallel auxiliary data structure that does not change the underlying relation's representation
- An index takes time to generate, but speeds up queries afterwards
  - Equality predicates are faster
  - Range predicates are faster if the index is implemented using a B+ tree

3.17 Multi-Attribute Indexes

A multi-attribute index is an index on the concatenation of the attributes

- `CREATE INDEX <index> ON <table>(<attr1>, <attr2>, ...);`
- The order in which attributes are declared in the index matters
- A multi-attribute index speeds up queries on
  - `<attr1> = '...'`
  - `<attr1> = '...' AND <attr2> = '...'`
  - `<attr1> = '...' AND <attr2> = '...' AND ..`

  Any other combination of queries will not benefit from the index. This includes the following
  - `<attr2> = '...' AND <attr1> = '...'`
  - `<attr2> = '...'`

4 Concurrency

4.1 Concurrency Control

Concurrency control is the process of deciding how to interleave operations

4.2 Concurrency Control Read Problems

- Inconsistent read
  - A dirty/inconsistent read happens when data is read during a write
  - This is a write-read (WR) conflict
- Unrepeatable read
  - An unrepeatable read happens when data is read prior to an update
  - This is a read-write (RW) conflict
- Phantom read
  - A phantom read happens when data is read prior to an addition/deletion
  - This is a read-write (RW) conflict

4.3 Concurrency Control Write Problems

- Lost update
  - A lost update happens when a write gets overwritten by another write
  - This is a write-write (WW) conflict

4.4 ACID Consistency Model

ACID is a set of principles that an ideal DBMS follows, although it is common to sacrifice some of these for performance gains

- Atomicity
  An operation on a database is atomic if it can only be executed as a whole
- Consistency
  An operation on a valid database is consistent if the database is still valid after the operation is executed
- Isolation
  An operation on a database is isolated if it behaves as if it is the only one running on the system
- Durability
  An operation on a database is durable if the updates persist even after the application terminates

A DBMS ensures atomicity and isolation through transactions and serializable schedules of transactions respectively

4.5 Transactions

A transaction is a sequence of operations on the database which are executed together, as if they were a single atomic operation. There are two ways to define transactions in SQL

- Each SQL command is a transaction
- ```
  BEGIN TRANSACTION
    <SQL Statement 1>
    <SQL Statement 2>
    ...
  COMMIT/ROLLBACK;
  ```
  - `COMMIT` finalizes execution
  - `ROLLBACK` undoes all changes made during execution

4.6 Schedules

A schedule is a sequence of interleaved operations from all transactions

- The following notation is used for the operations in transaction i
  - $\mathrm{R}_i(A)$ reads element A
  - $\mathrm{W}_i(A)$ updates element A
  - $\mathrm{I}_i(A)$ inserts/creates an element A
  - $\mathrm{D}_i(A)$ deletes an element A

4.7 Serial Schedules

A serial schedule is one where all operations of one transaction are performed to completion before the next one is started

- Advantages:
  - There are no concurrency control problems in a serial schedule
- Disadvantages:
  - When a transaction is waiting for I/O, other transactions cannot proceed
  - Multi-core hardware cannot be used

4.8 Serializable Schedules

A serializable schedule is one where transactions are executed concurrently and seemingly in isolation

- The DBMS state after transactions in a serializable schedule is identical to that after transactions in a serial schedule
  ```

4.9 Conflicts

A conflict between two operations occurs when changing their order might result in a different DBMS state

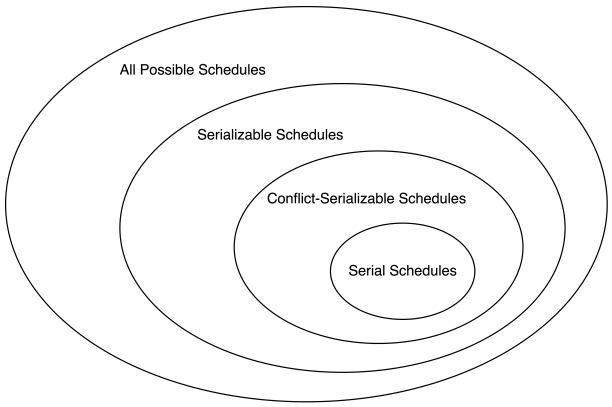
- Intra-transaction conflicts
  - Pairs of operations within a single transaction cannot be reordered without more info
- Inter-transaction conflicts
  - Pairs of operations between multiple transactions cannot be reordered without more info
  - Includes WR, RW, and WW conflicts

4.10 Conflict-Serializable Schedules

A conflict-serializable schedule can be reordered into a serial schedule by repeatedly swapping non-conflicting operations

- Conflict-serializability implies serializability
- Serializability does not imply conflict-serializability

4.11 Schedules



- Serializable schedules are those with the same final state as serial schedules
- Conflict-serializable schedules are those which can be reordered into serial schedules under conflict rules

4.12 Verifying Conflict-Serializability

```
function boolean isConflictSerializable(transactions) {
    for T in transactions {
        graph.insertVertex(T);
    }
    for T in transactions {
        for T' in transactions - T {

            for operation x in T {
                for operation y in T' {
                    if x.conflictsWith(y) {
                        graph.insertEdge(T, T');
                    }
                }
            }
        }
    }
    return graph.containsCycle();
}
```

- The resulting graph is called a precedence graph

4.13 Locks

A lock is a concurrency control technique that allows at most one thread to own it at a time

- Each element in a database has a unique lock
- A transaction must **acquire** an element's lock before reading/writing to that element
- A transaction must **release** an element's lock once it is done with that element
- The following notation is used for the operations in transaction i
  - $L_i(X)$ acquires the lock on element X
  - $U_i(X)$ releases the lock on element X

4.14 Two-Phase Locking

Two-phase locking is a protocol where locks are acquired and released in two phases

- Expanding phase: locks are acquired and no locks are released
- Shrinking phase: locks are released and no locks are acquired

Characteristics of two-phase locking

- Two-phase locking guarantees serializability and therefore isolation
- Two-phase locking does not guarantee recoverability or valid rollbacks

4.15 Strict Two-Phase Locking

Strict two-phase locking is a variant of two-phase locking where all unlock requests must happen after the `COMMIT` / `ROLLBACK` statement

- Strict two-phase locking guarantees recoverability and valid rollbacks

4.16 Shared/Exclusive Locks

Shared/exclusive locks have 3 possible states

- Exclusive/Write $X_i(A)$
  - At most one thread may possess a lock at this state
- Shared/Read $S_i(A)$
  - Multiple threads may posses a lock at this state
- Unlocked $U_i(A)$
  - No threads posses a lock at this state

4.17 Lock Granularity

- Fine-grained
  Fine-grained locking locks small pieces of data, i.e. individual tuples
  - High concurrency
  - High overhead in managing locks
  - Can be implemented with simple binary locks
- Coarse-grained
  Coarse-grained locking locks large pieces of data, i.e. tables or an entire database
  - Low concurrency
  - Less overhead in managing locks
  - Need to escalate the restrictiveness of the lock depending on operation

5 Relational Algebra

5.1 Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields new instances of relations as output

- Relational algebra is a formal mathematical language used to represent execution plans

5.2 Relational Algebra Operators

- $\bowtie$ Join
- $⟕$ Left outer join
- $\times$ Cartesian product
- σ Selection
- π Projection
- $-$ Difference
- $\cup$ Union
- $\cap$ Intersection
- γ Grouping and aggregation
- τ Sort
- δ De-duplication
- ρ Rename

- Projection
  Projection is a unary operator that removes unspecified columns
  - Projection is analogous to `SELECT`
  - i.e. $\pi_{T.A,T.B}$ returns the A and B attributes of T
- Selection
  Selection is a unary operator that returns all tuples that satisfy a specified condition
  - Selection is analogous to `WHERE` and `HAVING`
  - i.e. $\sigma_{T.A<6}$ returns all tuples in T where $A < 6$
- Difference
  Difference is a binary operator that returns all tuples in the first table that are not in the second
  - Selection is analogous to `EXCEPT`
  - i.e. $T_1 - T_2$ returns all tuples in T_1 that are not in T_2
- Grouping
  Grouping is a unary operator that groups the specified attributes and then aggregates them according to the specified function
  - Grouping is analogous to `GROUP BY`
  - i.e. $\gamma_{T.A,\max(T.B)}$ groups tuples by attribute A and then aggregates attribute B according to the `MAX` function
- Sort
  Sort is a unary operator that sorts tuples by the specified columns
  - Sort is analogous to `ORDER BY`
  - i.e. $\tau_{T.A,T.B}$ sorts tuples in ascending order of attribute A first, and then attribute B

- De-duplication
  De-duplication is a unary operator that removes all duplicate tuples
  - De-duplication is analogous to `GROUP BY` on all attributes
  - i.e. $\delta(T)$ returns all distinct tuples in T
- Inner join
  Inner join is a binary operator that joins two tables based on the specified condition
  - Inner join is analogous to `JOIN`
  - i.e. $\bowtie_{T_1.A=T_2.B}$ returns all (T_1, T_2) tuple pairs where $T_1.A = T_2.B$
- Natural join
  Natural join is a binary operator that joins two tables based on all columns with the same name in both tables
  - i.e. $T_1(A, B) \bowtie T_2(B, C)$ returns $R(A, B, C)$ where $T_1.B = T_2.B$
  - i.e. $T_1(A, B, C) \bowtie T_2(B, C, D)$ returns $R(A, B, C, D)$ where $(T_1.B, T_1.C) = (T_2.B, T_2.C)$
  - i.e $T_1(A, B) \bowtie T_2(C, D)$ returns $R(A, B, C, D)$, where $R = T_1 \times T_2$
  - i.e. $T_1(A, B) \bowtie T_2(A, B)$ returns $R(A, B)$, where $R = T_1 \cap T_2$
- Cartesian product
  Cartesian product is a binary operator that joins two tables
  - Cartesian product is analogous to `JOIN` without a join predicate
  - i.e. $T_1 \times T_2$ returns all (T_1, T_2) tuple pairs

5.3 Query Plan

A query plan is an expression in relation algebra, usually written as a tree

- The DBMS translates SQL queries into a query plan
- The plan specifies the order of operators

5.4 Query Optimization

- Only the last projection matters
- Inner-join order does not matter

5.5 Physical Operators

Physical operators describe the algorithm used to implement logical operators, i.e. the operators used in relational algebra

- Main memory operators
  - Assume that the data fits in main memory
  - Best for performance but may be limited as to how much data they can process
  - Performance is measured in terms of CPU instructions
- External memory algorithms
  - Assume that the data is on disk
  - Slower but no limitations on data size
  - Performance is measured in terms of loaded disk blocks

5.6 External Memory Algorithms

$B(R)$ denotes the number of blocks and $T(R)$ denotes the number of tuples in relation R

- Typically $B(R) < T(R)$

5.7 Naive Nested-Loop Join

```
for tuple tR in R:
    for tuple tS in S:
        if canJoin(tR, tS):
            output(tR, tS)
```

- Naive nested-loop join loads $T(R) \cdot B(S)$ disk blocks

5.8 Block-at-a-time Nested-Loop Join

```
for block bR in R:
    for block bS in S:

        for tuple tR in bR:
            for tuple tS in bS:
                if canJoin(tR, tS):
                    output(tR, tS)
```

- Block-at-a-time nested-loop join loads $B(R) + B(R)B(S)$ disk blocks

5.9 Block Nested-Loop Join

```
for group of N blocks nbR in R:
    for block bS in S:

        for tuple tR in nbR:
            for tuple tS in bS:
                if canJoin(tR, tS):
                    output(tR, tS)
```

- Block nested-loop join loads $B(R) + \lceil B(R)/N \rceil B(S)$ disk blocks

5.10 Hash Join

```
hashTable = new HashTable()

// build phase
for block bS in S:
    for tuple tS in bS:
        hashTable.add(tS)

// probe phase
for block bR in R:
    for tuple tR in bR:
        for tS in hashTable.find(tR):
            output(tR, tS)
```

- Assume that S is the smaller table
- Hash join loads $B(R) + B(S)$ disk blocks

5.11 Indexes

An index is a separate file that allows direct access to a row based on attributes

5.12 B+ Trees

A B+ tree is an m-ary tree with a large number of children per node that can be used to implement indexes. It consists of

- Internal nodes
  Internal nodes contain a subset of keys to serve as signposts to find the correct leaf
- Leaf nodes
  Leaf nodes contain key-value pairs and a pointer to the next leaf node to enable fast range queries

Characteristics of B+ trees

- B+ tree nodes take up approximately one disk block
- B+ tree nodes maintain $\sim 50\%$ occupancy of keys

5.13 Clustered Indexes

A clustered index is an index whose tuples are sorted in disk in the same order as the index's key order

- Typically there is only one clustered index per table

5.14 Unclustered Indexes

An unclustered index is an index whose tuples are sorted in disk in some arbitrary order

- There can be multiple unclustered indexes per table

5.15 Index-Based Selection

Given a relational algebra selection on table R and attribute A, there are multiple approaches to finding all tuples that satisfy the specified condition

- Approach 1
  - Perform a full table sequential scan
  - Selection on-the-fly
  - Loads $B(R)$ disk blocks
- Approach 2
  - Use a clustered index to look up records that satisfy the specified condition
  - Loads $B(R)/V(R, A)$ disk blocks
- Approach 3
  - Use an unclustered index to look up records that satisfy the specified condition
  - Loads $T(R)/V(R, A)$ disk blocks

where $V(R, A)$ is the number of distinct tuples in table R

5.16 Index Joins

```
for tuple tR in R:
    for tS in index.find(tR):
        output(tR, tS)
```

- When joining $R \bowtie_{R.A=S.B} S$, index join requires an index on $S.B$
- Clustered index join loads $T(R)\,(B(S)/V(S,B))$ disk blocks
- Unclustered index join loads $T(R)\,(T(S)/V(S,B))$ disk blocks

6 Parallel Processing

6.1 Workload Types

- Online Transaction Processing (OLTP)
  - No guarantees on technical or domain-specific knowledge
  - Queries typically read/update single records
- Online Analytical Processing (OLAP)
  - Data analyst or data scientist; few users, usually insiders
  - Queries typically touch most of the data

6.2 Intra-Query Parallelism

In intra-query parallelism, a single query is decomposed into smaller tasks that execute concurrently on multiple processors

- Also known as intra-operator parallelism
- Scales well for complex analytical queries

6.3 Inter-Query Parallelism

In inter-query parallelism, several different queries execute concurrently on multiple processors

- Also known as inter-operator parallelism
- Scales well for large volumes of simple transactions

6.4 Data Replication

In data replication, copies of the dataset are shared across computing nodes

- Expensive
- OLAP workloads unlikely to cause consistency issues
- Commonly used for intra-query parallelism

6.5 Data Partitioning

In data partitioning, the dataset is split across computing nodes

- Difficult to avoid consistency issues
- Can accommodate OLTP workloads but rarely compatible with OLAP workloads
- Commonly used for inter-query parallelism

6.6 Data Partitioning Methods

- Horizontal partitioning
  In horizontal partitioning, every attribute is available but not every tuple is present
  - Block partitioning
    - Tuples are partitioned by raw size of partition
  - Range partitioning
    - Each partition contains tuples with a range of attribute values
  - Hash partitioning
    - Each partition contains tuples with a range of attribute hashes
- Vertical partitioning
  In vertical partitioning, every tuple value of an attribute is available on the replica but not every attribute is present

6.7 Partition Quality

- Uniform partition
  In a uniform partition, all partitions are roughly the same size
  - Block partitioning guarantees uniform partitions
- Skewed partition
  In a skewed partition, some partitions may be significantly larger than others
  - Range and hash partitioning may produce skewed partitions

6.8 Shared-Nothing Intra-Operator Database

A shared-nothing intra-query database consists of nodes which are on a common network and may carry out specified relational operations

- Partitioned selection
  In a partitioned selection, each node computes the selection independently and the results are unioned at the end
  - If the data is hash-partitioned on the selected attribute, then only one node computes the selection
- Partitioned aggregation
  In a partitioned aggregation, attributes are hashed such that tuples with the same attributes are sent to the same node. The node then aggregates the tuples and the results are unioned at the end
  - Tuples may be aggregated before being sent to their corresponding nodes to reduce the number of nodes sent across the network
- Partitioned join
  In a partitioned join, tuples are hashed on their join attributes such that tuples with equal join attributes are sent to the same node. The node then joins the tuples and the results are unioned at the end
  - Partitioned join is susceptible to skew
- Broadcast join
  In a broadcast join, an unpartitioned table is joined with a partitioned table. The former is broadcast to every node and each node joins the former with the latter. The results are unioned at the end

7 NoSQL Systems

7.1 BASE Consistency Model

BASE is a set of principles that an ideal NoSQL system follows

- Basically Available
  Basic features work most of the time and most failures do not cause a full system outage
- Soft state
  System is not always write consistent and replicas do not have to be mutually consistent
- Eventually consistent
  System will eventually be consistent

7.2 Brewer's Conjecture

A distributed data store can only provide two of the following three guarantees

- Consistency
  Every read receives the most recent write or an error
- Availability
  Every request must respond with a non-error
- Partition tolerance
  Continued operation in the presence of dropped or delayed messages

7.3 RDBMS & NoSQL Systems

- Relational database management systems (RDBMS) are intended to be highly consistent
  - Can boost availability by sacrificing some consistency
- NoSQL systems are intended to be highly available
  - Can boost consistency by sacrificing some availability

7.4 Key-Value Stores

Key-value stores represent data as `(key, value)` pairs in a hash table

- Basic operations:
  - `get(key)`
  - `put(key, value)`
- Distribution/Partitioning:
  - Access via hash function
  - Replication:
    - None: key k stored at server $h(k)\%N$
    - 3-way: key k stored at servers $h_1(k)\%N,\ h_2(k)\%N,\ h_3(k)\%N$

7.5 Document Stores

Document stores represent data as a hash table over semi-structured documents

- A semi-structured document is any parseable file containing explicit tags to indicate metadata such as type or name
  - i.e. JSON files

7.6 SQL++

Structured Query Language++ (SQL++) is a language used to query semi-structured data

- SQL++ is syntactically similar to SQL but operates on a different data model

7.7 SQL++ Types

SQL++ types define the reusable schema of a collection

- All fields have a name and an associated type
  - By default every field is required in the document
  - Optional fields are specified with "`?`"
- Types are specified as either `OPEN` or `CLOSED`
  - `OPEN` indicates that additional fields may be added to the type
  - `CLOSED` indicates that no other fields are allowed
- A field has uniform type if all data are of the same type
  - Uniformity can be coerced by converting all data into a single type
    - Type identification functions allow different types to be converted in different ways
    - i.e. `IS_ARRAY(...), IS_BOOLEAN(...)`

7.8 Graph Stores

Graph stores represent data as a directed graph of entities (vertices) and relationships (directed edges)

- Entities and relationships can have types and attributes

7.9 Cypher

Cypher is a language used to query graph-stored data

- Cypher is used by the Neo4j graph database management system

7.10 Cypher Queries

`MATCH` in Cypher is analogous to a combined `FROM/WHERE` in SQL

- `MATCH (nodeName:TYPE {propertyName:'propertyValue'})`
  - Matches a node with the desired label and property value
- `MATCH -[edgeName:TYPE]->`
  - Matches an edge with the desired type
- `MATCH (nodeName:TYPE)-[edgeName:TYPE]->(nodeName:TYPE)`
  - Matches two nodes and an edge