Normalization

Spring 2024

What is Normalization…

- Normalization is the process of organizing data in a database

- It includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency

Why Normalization…

- Reduce data redundancy
- Increase data consistency
- Improve database performance
- Protect data
- Make the database more flexible

Why Normalization…

- **Normalization is basically to design a database schema such that duplicate and redundant data is avoided.**

- **If the same information is repeated in multiple places in the database, there is the risk that it is updated in one place but not the other, leading to data corruption.**

- **Interview Video Link: https://www.youtube.com/watch?v=SEdAF8mSKS4**

Anomalies

- **Update Anomaly**

- Let say we have 10 columns in a table
- 2 columns are called employee Name and employee address.
- Now if one employee changes location then we would have to update the table.
- But the problem is, if the table is not normalized one employee can have multiple entries and while updating all of those entries one of them might get missed.

Anomalies

- **Insertion Anomaly**

- **Let’s say we have a table that has 4 columns. Student ID, Student Name, Student Address and Student Grades.**

- **Now when a new student enroll in school, even though first three attributes can be filled but 4th attribute will have NULL value because he doesn't have any marks yet.**

Anomalies

- **Deletion Anomaly**

- **This anomaly indicates unnecessary deletion of important information from the table.**
- **Let's say we have student's information and courses they have taken as follows (student ID,Student Name, Course, address).**
- **If any student leaves the school then the entry related to that student will be deleted.**
- **However, that deletion will also delete the course information even though course depends upon the school and not the student.**

Deletion Anomaly

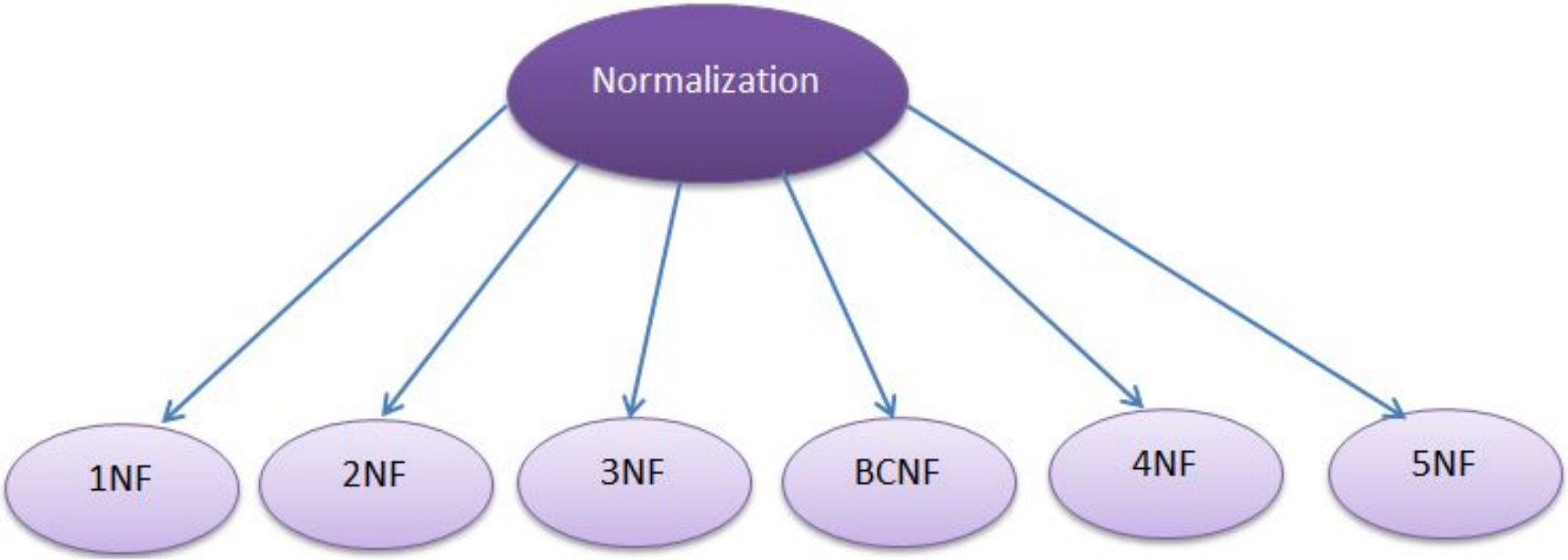
- **For example, if we try to delete the student Subbu:**
  - **we will lose the information that R. Prasad teaches C.**
  - **These difficulties are caused by the fact the teacher is determinant but not a candidate key.**

Student	Teacher	Subject
Jhansi	P.Naresh	Database
jhansi	K.Das	C
subbu	P.Naresh	Database
subbu	R.Prasad	C

Con of Normalization…

- Normalization divides the larger table into smaller and links them using relationships.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

Normal Forms



1st Normal Form

- **First Normal Form (1NF):** This is the most basic level of normalization.
- In 1NF, each table cell should contain only a single value, and each column should have a unique name.
- The first normal form helps to eliminate duplicate data and simplify queries.
- If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute.
- A relation is in first normal form if every attribute in that relation is singled valued attribute.

1st Normal Form

A table is in the first normal form iff:

- **Rule 1:** The domain of each attribute contains only atomic values
- **Rule 2:** The value of each attribute contains only a single value from that domain.

In layman's terms. it means every column of your table should only contain *single values*

1st Normal Form Example

For a Library: Not in 1NF

Patron ID	Borrowed books
C45	B33, B44, B55
C12	B56

1st Normal Form Example

For a Library: 1NF Solution

Patron ID	Borrowed book
C45	B33
C45	B44
C45	B33
C12	B56

1st Normal Form Example

For a Airline: Not in 1NF

Flight	Weekdays
UA59	Mo We Fr
UA73	Mo Tu We Th Fr

1st Normal Form Example

For a Airline: 1NF Solution

Flight	Weekday
UA59	Mo
UA59	We
UA59	Fr
UA73	Mo
UA73	We

Functional Dependency

- **Functional dependencies** are relationships between attributes in a database.
- They describe how one attribute is dependent on another attribute.
- For example:
- Consider a database of employee records. The employee's ID number might be functionally dependent on their name because the name determines the ID number. In this case, we would say that the ID number is functionally dependent on the name.

Functional Dependency

- In a relational database management, functional dependency is a concept that specifies the relationship between two sets of attributes where one attribute determines the value of another attribute.

- It is denoted as X → Y
  - where the attribute set on the left side of the arrow, **X is called Determinant, and Y is called the Dependent.**

roll_no	name	dept_name	dept_building

- roll_no → { name, dept_name, dept_building }
  - Here, roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency

Functional Dependency

roll_no	name	dept_name	dept_building

- **roll_no → { name, dept_name, dept_building }**
  - **Here, roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency**

- **roll_no → dept_name , Since, roll_no can determine whole set of {name, dept_name, dept_building}, it can determine its subset dept_name also.**

- **dept_name → dept_building , Dept_name can identify the dept_building accurately, since departments with different dept_name will also have a different dept_building**

- **More valid functional dependencies: roll_no → name, {roll_no, name} → {dept_name, dept_building}, etc.**

Revision

- **Candidate Key:** A Candidate Key can be any column or a combination of columns that can qualify as unique key in database.
- There can be multiple Candidate Keys in one table.
- Each Candidate Key can qualify as Primary Key.
- Candidate key is the minimal super key
- **Primary Key:** A Primary Key is a column or a combination of columns that uniquely identify a record.
- Only one Candidate Key can be Primary Key.

2nd Normal Form

A table is in 2NF:

- **It is in 1NF**
- **It must not have any partial dependencies.**

- A partial dependency occurs when a non-key attribute is dependent on only a part of the primary key.

- In other words, the value of the **non-key/non-prime** attribute can be determined by a subset of the primary key but not the entire key.

- Partial dependency occurs when a table has redundant data, which can lead to problems such as data inconsistency, data anomalies, and poor performance.

2nd Normal Form

- **Second Normal Form (2NF):** To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency.

- A relation is in 2NF if it has **No Partial Dependency**, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

- **Partial Dependency** – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

2nd Normal Form

BookNo	Patron	PhoneNo
B3	J. Fisher	555-1234
B2	J. Fisher	555-1234

- **Candidate key is {BookNo, Patron}**
- **We have Patron → PhoneNo**
  - **PhoneNo is a non-prime attribute ⇒ Not in 2NF**

- **It is already in 1 NF**

BookNo	Patron
B3	J. Fisher
B2	J. Fisher
B2	M. Amer

Patron	PhoneNo
J. Fisher	555-1234
M. Amer	555-4321

Example (Table violates 2NF)

<StudentProject>

StudentID	ProjectID	StudentName	ProjectName
S89	P09	Olivia	Geo Location
S76	P07	Jacob	Cluster Exploration
S56	P03	Ava	IoT Devices
S92	P05	Alexandra	Cloud Deployment

In the above table, we have partial dependency; let us see how –

The prime key attributes are **StudentID** and **ProjectID**.

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent.

The **StudentName** can be determined by **StudentID**, which makes the relation Partial Dependent.

The **ProjectName** can be determined by **ProjectID**, which makes the relation Partial Dependent.

Therefore, the <**StudentProject**> relation violates the 2NF in Normalization and is considered a bad database design.

<StudentInfo>

StudentID	ProjectID	StudentName
S89	P09	Olivia
S76	P07	Jacob
S56	P03	Ava
S92	P05	Alexandra

<ProjectInfo>

ProjectID	ProjectName
P09	Geo Location
P07	Cluster Exploration
P03	IoT Devices
P05	Cloud Deployment

2nd Normal Form

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

- non-prime attribute TEACHER_AGE is dependent on TEACHER_ID
- TEACHER_ID which is a proper subset of a candidate key
- That's why it violates the rule for 2NF

2nd Normal Form

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

3rd Normal Form

- **Third Normal Form (3NF):** A relation is said to be in third normal form, if we did not have any transitive dependency for non-prime attributes.

- **The basic condition with the Third Normal Form is that, the relation must be in Second Normal Form.**

A table is in 3NF:

- **It is in 2NF and**
- **all its attributes are determined only by its candidate keys and not by any non-prime attributes**

3rd Normal Form

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

3rd Normal Form

- **Transitive Dependency**
  - **EMP_STATE & EMP_CITY dependent on EMP_ZIP**
  - **EMP_ZIP dependent on EMP_ID**

The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID)

3rd Normal Form

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

In the table able, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE_GENRE], [Genre Type] is only dependent on [Genre ID].

BCNF

BCNF (Boyce-Codd Normal Form): It is just a advanced version of Third Normal Form.

Here we have some additional rules than Third Normal Form. The basic condition for any relation to be in BCNF is that it must be in Third Normal Form.

We have to focus on some basic rules that are for BCNF:

1. Table must be in Third Normal Form.
2. In relation X->Y, X must be a key in a relation.

BCNF

- BCNF is free from redundancy caused by Functional Dependencies.
- If a relation is in BCNF, then 3NF is also satisfied.
- Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
- There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it’s generally not required to go beyond BCNF

BCNF

Stu_ID	Stu_Branch	Stu_Course	Branch_Number	Stu_Course_No
101	Computer Science & Engineering	DBMS	B_001	201
101	Computer Science & Engineering	Computer Networks	B_001	202
102	Electronics & Communication Engineering	VLSI Technology	B_003	401
102	Electronics & Communication Engineering	Mobile Communication	B_003	402

Functional Dependency of the above is as mentioned:

```
Stu_ID -> Stu_Branch
Stu_Course -> {Branch_Number, Stu_Course_No}
```

Candidate Keys of the above table are: **{Stu_ID, Stu_Course}**

BCNF Example

- **Not in BCNF:**
  - **neither Stu_ID nor Stu_Course is a Super Key**

- **Break into more tables !!**

Stu_Branch Table

Stu_ID	Stu_Branch
101	Computer Science & Engineering
102	Electronics & Communication Engineering

Candidate Key for this table: **Stu_ID**.

Stu_Course Table

Stu_Course	Branch_Number	Stu_Course_No
DBMS	B_001	201
Computer Networks	B_001	202
VLSI Technology	B_003	401
Mobile Communication	B_003	402

Candidate Key for this table: **Stu_Course**.

BCNF Example

Student	Teacher	Subject
Jhansi	P.Naresh	Database
jhansi	K.Das	C
subbu	P.Naresh	Database
subbu	R.Prasad	C

F: { (student, Teacher) -> subject
(student, subject) -> Teacher
Teacher -> subject}

Candidate keys are (student, teacher) and (student, subject).

BCNF Example

R1

Teacher	Subject
P.Naresh	database
K.DAS	C
R.Prasad	C

R2

Student	Teacher
Jhansi	P.Naresh
Jhansi	K.Das
Subbu	P.Naresh
Subbu	R.Prasad

BCNF Example

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

BCNF Example

EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Concurrency Control

Spring 2024

Conflict Serializable Schedules

- **Two schedules are conflict equivalent if:**
  - **Involve the same actions of the same transactions**
  - **Every pair of conflicting actions is ordered the same way**

- **Schedule S is conflict serializable if S is conflict equivalent to some serial schedule**

Dependency Graph

- **Dependency graph:**
  - **One node per Xact; edge from Ti to Tj if Tj reads/writes an object last written by Ti.**

- **Theorem: Schedule is conflict serializable if and only if its dependency graph is acyclic.**

Conflict Serializable Schedules

Step 1: Draw a node for each transaction in the schedule.

Step 2: For each pair of conflicting operations (i.e., operations on the same data item by different transactions), draw an edge from the transaction that performed the first operation to the transaction that performed the second operation. The edge represents a dependency between the two transactions.

Step 3: If there are multiple conflicting operations between two transactions, draw multiple edges between the corresponding nodes.

Step 4: If there are no conflicting operations between two transactions, do not draw an edge between them.

Step 5: Once all the edges have been added to the graph, check if the graph contains any cycles. If the graph contains cycles, then the schedule is not conflict serializable. Otherwise, the schedule is conflict serializable.

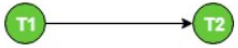
Conflict Serializable Schedules

Step 1: Make two nodes corresponding to Transaction T_1 and T_2.

Step 1

Step 2: For the conflicting pair r1(x) w2(x), where r1(x) happens before w2(x), draw an edge from T_1 to T_2.

```
S: r1(x) r1(y) w2(x) w1(x) r2(y)
```



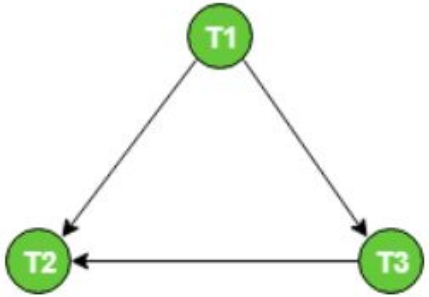
Step 2

Step 3: For the conflicting pair w2(x) w1(x), where w2(x) happens before w1(x), draw an edge from T_2 to T_1.

Step 3

Conflict Serializable Schedules

```
S1: r1(x) r3(y) w1(x) w2(y) r3(x) w2(x)
```



Precedence Graph

Example

- A schedule that is not conflict serializable:

T1: R1(A), W(A) R(B), W(B)

T2: R(A), W(A), R(B), W(B)

- The cycle in the graph reveals the problem. The output of T1 depends on T2, and vice versa.

Review: Strict 2PL

- **Strict Two-phase Locking (Strict 2PL) Protocol:**
  - **Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.**
  - **All locks held by a transaction are released when the transaction completes**
  - **If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.**

- **Strict 2PL allows only schedules whose precedence graph is acyclic**

Two Phase Locking (2PL)

- **Two-Phase Locking Protocol:**
  - **Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.**
  - **A transaction can not request additional locks once it releases any locks.**
  - **If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.**

Lock Management

- Lock and unlock requests are handled by the lock manager
- Lock table entry:
  - Number of transactions currently holding a lock
  - Type of lock held (shared or exclusive)
  - Pointer to queue of lock requests
- Locking and unlocking have to be atomic operations
- Lock upgrade: transaction that holds a shared lock can be upgraded to hold an exclusive lock

Deadlocks

- **Deadlock: Cycle of transactions waiting for locks to be released by each other.**

- **Two ways of dealing with deadlocks:**
  - **Deadlock prevention**
  - **Deadlock detection**

Deadlock Prevention

- Assign priorities based on timestamps. Assume Ti wants a lock that Tj holds. Two policies are possible:
  - Wait-Die: It Ti has higher priority, Ti waits for Tj; otherwise Ti aborts
  - Wound-wait: If Ti has higher priority, Tj aborts; otherwise Ti waits

- If a transaction re-starts, make sure it has its original timestamp

Deadlock Prevention

Wait – Die	Wound -Wait
It is based on a non-preemptive technique.	It is based on a preemptive technique.
In this, older transactions must wait for the younger one to release its data items.	In this, older transactions never wait for younger transactions.
The number of aborts and rollbacks is higher in these techniques.	In this, the number of aborts and rollback is lesser.

Storage and Modern Databases

Spring 2024

Types of Databases

- **Small** : On Laptop (Local storage)
- **Large :** A large mainframe computer
- **Very Large:** Data Warehouses

- **Data about the data is called Metadata !!**

Types of Data Storage

- **Disks:** Can retrieve random page at fixed cost But reading several consecutive pages is much cheaper than reading them in random order
- **Tapes:** Can only read pages in sequence Cheaper than disks; used for archival storage
- **File organization:** Method of arranging a file of records on external storage.
  - Record id (rid) is sufficient to physically locate record
  - Indexes are data structures that allow us to find the record ids of records with given values in index search key fields
- **Architecture:** Buffer manager stages pages from external storage to main memory buffer pool. File and index layers make calls to the buffer manager

Alternative File Organization

Many alternatives exist, each ideal for some situations, and not so good in others:

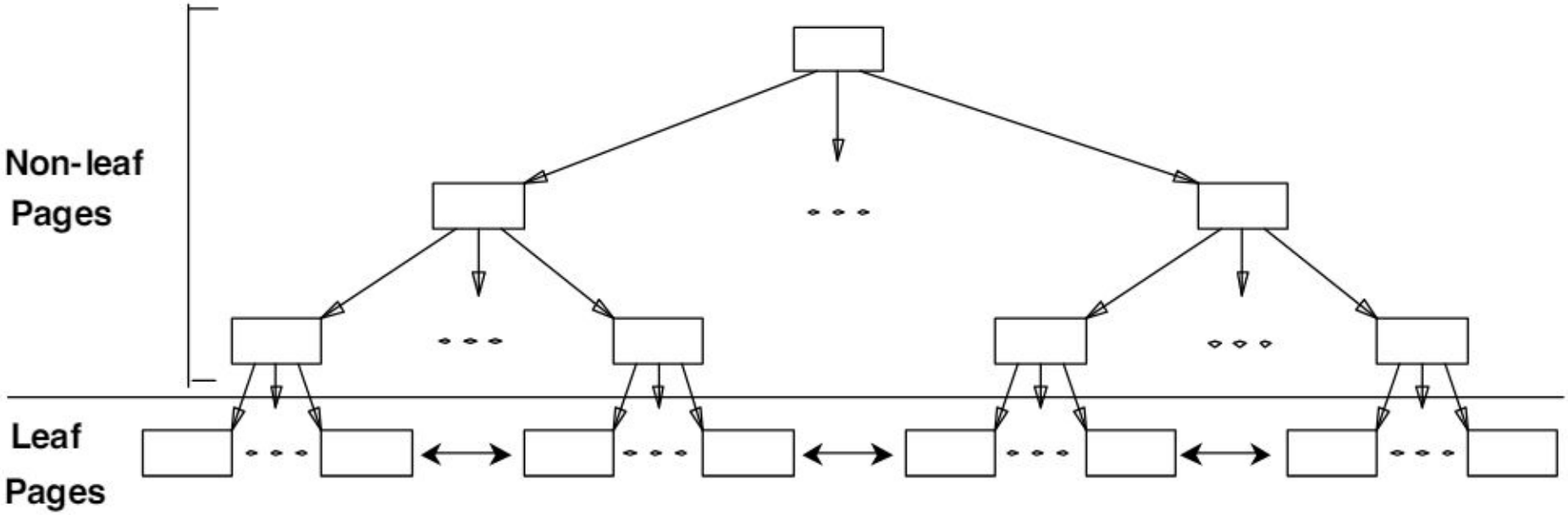
- **Heap** (random order) files: Suitable when typical access is a file scan retrieving all records.
- **Sorted Files**: Best if records must be retrieved in some order, or only a `range' of records is needed.
- **Indexes**: Data structures to organize records via trees or hashing. • Like sorted files, they speed up searches for a subset of records, based on values in certain ("search key") fields • Updates are much faster than in sorted files.

Indexes

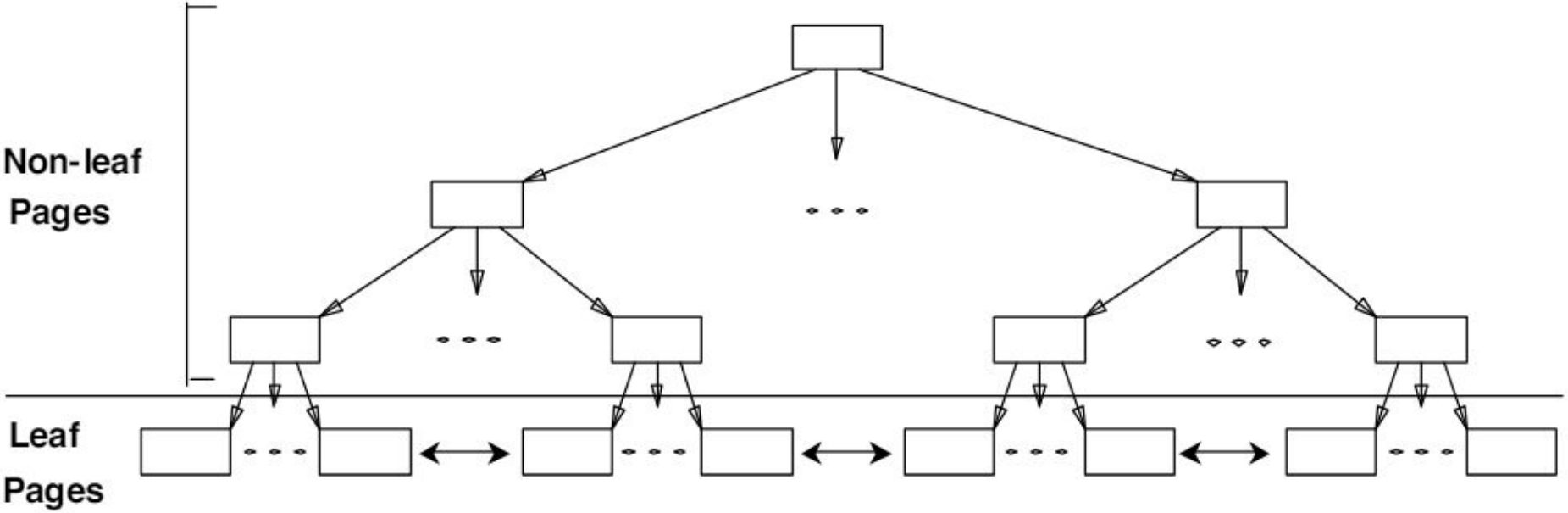
- An index on a file speeds up selections on the search key fields for the index.
  - Any subset of the fields of a relation can be the search key for an index on the relation.
  - Search key is not the same as key (minimal set of fields that uniquely identify a record in a relation).

- An index contains a collection of data entries, and supports efficient retrieval of all data entries k* with a given key value k.

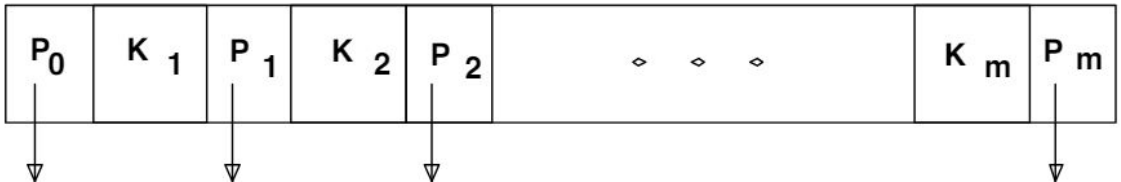
B+ Tree



B+ Tree



- Leaf pages contain data entries, and are chained (prev & next)
- Non-leaf pages contain index entries and direct searches:



Operations to Compare

- Scan: Fetch all records from disk
- Equality search
- Range selection
- Insert a record
- Delete a record

- Data entries can be actual data records, pairs, or pairs.
- Choice orthogonal to indexing technique used to locate data entries with a given key value
- Indexes must be chosen to speed up important queries (and perhaps some updates!)

Cloud Storage

- Cloud Storage uses remote servers to save data, such as files, business data, videos, or images. Users upload data to servers via an internet connection, where it is saved on a virtual machine on a physical server.

- To maintain availability and provide redundancy, cloud providers will often spread data to multiple virtual machines in data centers located across the world.

- If storage needs increase, the cloud provider will spin up more virtual machines to handle the load

Cloud Storage

- **B+ trees are an essential component of contemporary database systems since they significantly improve database performance and make efficient data management possible.**

- **Traditional B+ trees are not suitable for cloud environments. However, some papers present a scalable B+-tree based indexing scheme for efficient data processing in the cloud.**

- **AWS S3, you can store relational, hierarchical, semi-structured, or completely unstructured data.**

Modern Databases

- **MongoDB (NOSQL Databases): uses objectID**

- **Amazon AWS S3 : you can store relational, hierarchical, semi-structured, or completely unstructured data.**

- **Non-relational or NoSQL databases are also used to store data, but unlike relational databases, there are no tables, rows, primary keys, or foreign keys. Instead, these data-stores use models optimized for specific data types.**

    - **The four most popular non-relational types are document data stores, key-value stores, graph databases, and search engine stores.**

Modern Databases

- **Non-relational or NoSQL databases**
  - **Document data stores**
  - **Columnar (or column oriented) data stores**
  - **Key value stores**
  - **Document stores**
  - **Graph databases**

Modern Databases

Features of relational databases

- They work with structured data.
- Relationships in the system have constraints, which promotes a high level of data integrity.
- There are limitless indexing capabilities, which results in faster query response times.
- They are excellent at keeping data transactions secure.
- They provide the ability to write complex SQL queries for data analysis and reporting.
- Their models can ensure and enforce business rules at the data layer adding a level of data integrity not found in a non-relational database.
- They are table and row oriented.
- They Use SQL (structured query language) for shaping and manipulating data, which is very powerful.

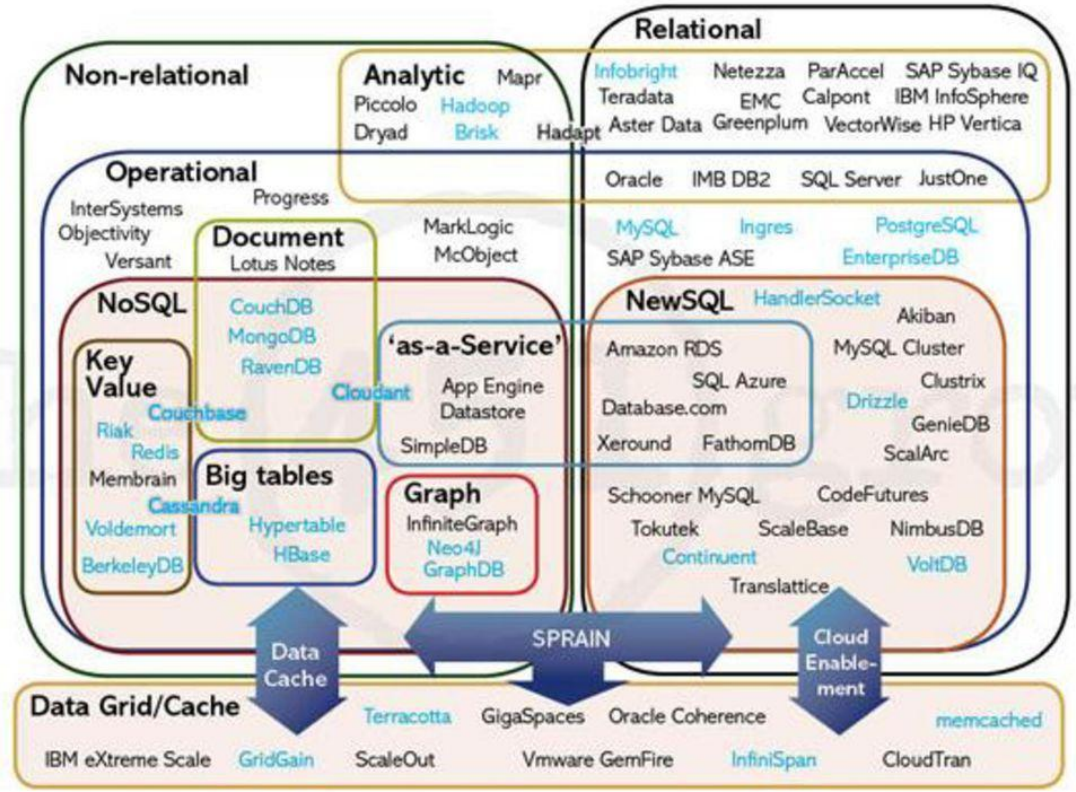
Features of non-relational databases

- They have the ability to store large amounts of data with little structure.
- They provide scalability and flexibility to meet changing business requirements.
- They provide schema-free or schema-on-read options.
- They have the ability to capture all types of data “Big Data” including unstructured data.
- They are document oriented.
- NoSQL or non-relational databases examples: MongoDB, Apache Cassandra, Redis, Couchbase and Apache HBase.
- They are best for Rapid Application Development. NoSQL is the best selection for flexible data storage with little to no structure limitations.

Modern Databases

MongoDB	Amazon S3
MongoDB is a document database that stores data in JSON-like documents.	Amazon S3 is an object storage service that stores data as objects within buckets.
It stores data as documents.	It stores data as objects.
Data is stored in BSON format.	Data is stored in binary format.
MongoDB stores data in collections.	Amazon S3 stores data in buckets.
MongoDB stores data in databases.	Amazon S3 stores data in regions.
MongoDB is a NoSQL database / Schemaless as well as a cloud storage service	Amazon S3 is a cloud storage service

Modern Databases



Modern Databases

Transaction Management

Spring 2024

ACID

- **Atomicity:** All actions in the Xact happen, or none happen.
- **Consistency:** If each Xact is consistent, and the DB starts consistent, it ends up consistent.
- **Isolation:** Execution of one Xact is isolated from that of other Xacts.
- **Durability:** If a Xact commits, its effects persist.

Transaction: An Execution of a DB

- **Concurrent execution of user programs is essential for good DBMS performance.**
  - **Because disk accesses are frequent, and relatively slow, it is important to keep the cpu humming by working on several user programs concurrently**

- **A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the database**

- **A transaction is the DBMS's abstract view of a user program: a sequence of reads and writes**

Concurrency

- Users submit transactions, and can think of each transaction as executing by itself
- Concurrency is achieved by the DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.
- Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins.
    - DBMS will enforce some ICs, depending on the ICs declared in CREATE TABLE statements
    - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).

- **Issues**: Effect of interleaving transactions, and crashes

Atomicity

- A transaction might commit after completing all its actions, or it could abort (or be aborted by the DBMS) after executing some actions.

- DBMS logs all actions so that it can undo the actions of aborted transactions

- DBMS ensures atomicity (all-or-nothing property) even if system crashes in the middle of a Xact.

Atomicity Example

```
T1:   BEGIN  A=A+100,   B=B-100   END
T2:   BEGIN  A=1.06*A,  B=1.06*B  END
```

- Intuitively, the first transaction is transferring $100 from B's account to A's account. The second is crediting both accounts with a 6% interest payment.

- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect must be equivalent to these two transactions running serially in some order

Atomicity Example

Consider a possible interleaving (*schedule*):

T1:	A=A+100,		B=B-100
T2:		A=1.06*A,	B=1.06*B

This is OK. But what about:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

The DBMS's view of the second schedule:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	

Scheduling Transactions

- **Serial schedule:** Schedule that does not interleave the actions of different transactions

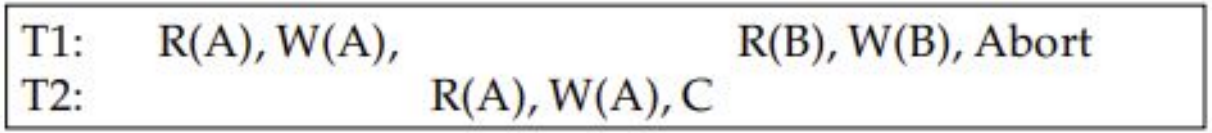
- **Equivalent schedules:** For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule

- **Serializable schedule:** A schedule that is equivalent to some serial execution of the transactions.

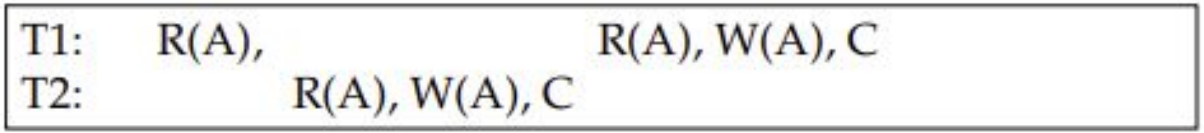
- Note: If each transaction preserves consistency, every serializable schedule preserves consistency.

Anomalies with Interleaved Transactions

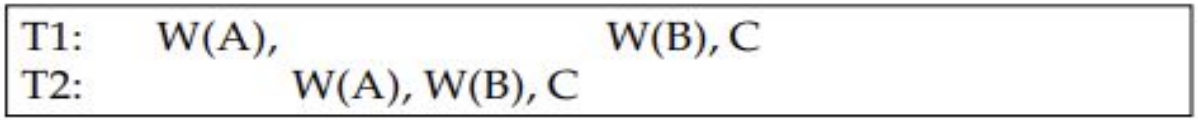
- **Reading Uncommitted Data (WR Conflicts,"dirty reads")**



- **Unrepeatable Reads (RW Conflicts)**



- **Overwriting Uncommitted Data (WW Conflicts)**



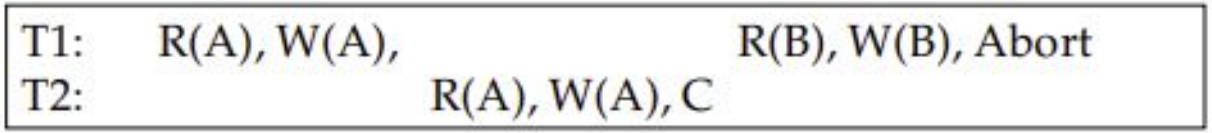
Lock-Based Concurrency Control

- **Strict Two-phase Locking (Strict 2PL) Protocol**
  - **Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing**
  - **All locks held by a transaction are released when the transaction completes**
  - **If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object**

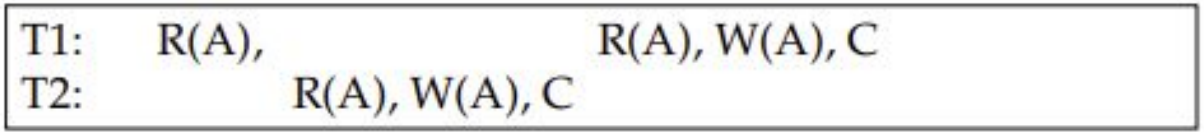
- **Strict 2PL allows only serializable schedules.**

Anomalies with Interleaved Transactions

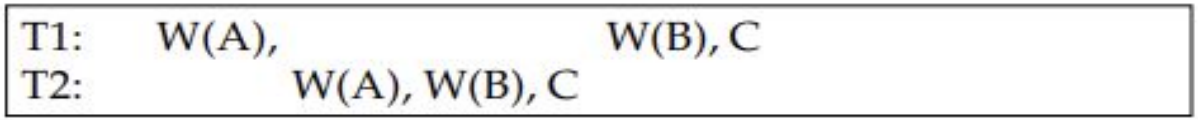
- **Reading Uncommitted Data (WR Conflicts,"dirty reads")**



- **Unrepeatable Reads (RW Conflicts)**



- **Overwriting Uncommitted Data (WW Conflicts)**



Aborting a Transactions

- If a transaction Ti is aborted, all its actions have to be undone. Not only that, if Tj reads an object last written by Ti, Tj must be aborted as well!

- Most systems avoid such cascading aborts by releasing a transaction's locks only at commit time.
    - If Ti writes an object, Tj can read this only after Ti commits

- In order to undo the actions of an aborted transaction, the DBMS maintains a log in which every write is recorded. This mechanism is also used to recover from system crashes: all active Xacts at the time of the crash are aborted when the system comes back up

The Log

- The following actions are recorded in the log:
  - Ti writes an object: the old value and the new value.
    - Log record must go to disk before the changed page!
  - T i commits/aborts: a log record indicating this action.
- Log records chained together by Xact id, so it's easy to undo a specific Xact (e.g., to resolve a deadlock).
- Log is often duplexed and archived on "stable" storage.
- All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

Recovering

- There are 3 phases in the Aries recovery algorithm:
  - **Analysis:** Scan the log forward (from the most recent checkpoint) to identify all Xacts that were active, and all dirty pages in the buffer pool at the time of the crash.
  - **Redo:** Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk
  - **Undo:** The writes of all Xacts that were active at the crash are undone (by restoring the before value of the update, which is in the log record for the update), working backwards in the log. (Some care must be taken to handle the case of a crash occurring during the recovery process!)

Summary

- **Concurrency control and recovery are among the most important functions provided by a DBMS.**

- **Users need not worry about concurrency.**
  - **System automatically inserts lock/unlock requests and schedules actions of different Xacts in such a way as to ensure that the resulting execution is equivalent to executing the Xacts one after the other in some order**

- **Write-ahead logging (WAL) is used to undo the actions of aborted transactions and to restore the system to a consistent state after a crash.**
  - **Consistent state: Only the effects of commited Xacts seen**