NOTES

00

Under the **write-ahead log protocol**, log records are written before the corresponding writes to the database. This protocol ensures the recovery manager can safely assume that if there's no *transaction commit* record in the log file for a particular transaction, then the transaction was still active at the time of failure and should be undone.

Checkpoints are used to limit the amount of searching and subsequent processing that we need to carryout on the log file during recovery.

002 + 034

The objectives of **query processing** are to:

1. *transform* a query written in a high-level language, like SQL, into a correct and efficient execution strategy expressed in a low-level language (implementing relational algebra), and to
2. *execute* the strategy to retrieve the required data.

004

A **timestamp** is a unique identifier created by the DBMS that indicates the relative starting time of a transaction. Timestamps are used in timestamp-based protocols for concurrency control to determine which transaction should get priority during a conflict.

006

A **transaction** is an action or series of actions carried out by a single user or application program which reads or updates the content of the database. A transaction is treated as one logical unit of work on the database

Transactions are important in DBMS because they enable us to ensure consistency and integrity of data in the database. It is also the unit of concurrency and recovery control.

008

Normalization is the technique of producing a set of relations with desirable properties, given the data requirements of an enterprise. It is a formal method that can be used to identify relations based on their keys and the functional dependencies among their attributes.

- 1NF: ensures the intersection of each row and columns contain just one value.
- 2NF: ensures all non-primary-key attributes are fully functionally dependent on the primary key
- 3NF: ensures no non-primary key attribute is transitively dependent on the primary key.

010

A **schedule** is a sequence of operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions.

- **A serial schedule** is a schedule in which the operations of each transaction are executed consecutively without any interleaved operations from other transactions.
- **A non-serial schedule** is a schedule in which the operations from a set of transactions are interleaved.
- **A serializable schedule** is a non-serial schedule that produces the same result as some serial execution.

012

The lock-based protocol is a procedure used to control concurrent access to data whereby a transaction accessing the database locks access to the data and therefore prevents other transactions from accessing incorrect results. Whereas in **a timestamp-based protocol**, the timestamps of transactions are used to determine which transaction has priority in the event of a conflict.

Important difference: In a timestamp method, no locks are used and therefore there can be **no deadlocks**. In the event of a conflict, there are no waiting and transactions involved are simply rolled back and restarted.

014 + 016

In **materialization**, the output of intermediate operations is stored in a temporary relation and are written to disk whereas in **pipelining**, the results of one operation are sent as the input to another operation without creating a temporary relation to hold the intermediate results. This is implemented as a separate thread or process within the DBMS, each pipeline takes a stream of inputs and creates a stream of output.

030

- **Atomicity**: The all or nothing property – a transaction is an indivisible unit that is either performed entirely, or not performed at all.
- **Consistency**: A transaction must transform a database from a consistent state to another consistent state.
- **Isolation**: Transactions execute independently of one another.
- **Durability**: the effects of a successfully completed transaction is permanently recorded in the database and must not be lost.

032

When two or more users are accessing the database simultaneously and at least one of them is updating data, there may be interferences that can result in inconsistencies.

- **Lost update problem**: An apparently successfully completed update operation by one user can be overridden by another user.
- **Uncommitted dependency** (dirty reads): occurs when one transaction can see the intermediate results of another transaction before it has committed. Problem arises if the uncommitted transaction is rolled back.
- **Inconsistent Analysis**: a transaction reads several values from the database but a second transaction updates some of them during the execution of the first.

036

5 types of **attributes** that represent an ER model

- Simple: e.g. salary
- Composite: e.g. address
- Single-valued: e.g. branch number
- Multivalued: e.g. phone number (can have more than one)
- Derived: e.g. age from DOB

038

Attributes represent properties of an entity or relationship type.

040

A **strong entity** is an entity which is not existence-dependent on another entity. It has its primary key and can be identified by that key. Whereas a **weak entity** is existence dependent on another entity type.

042

Path from thought to fulfilment:

The path to fulfilment starts from Thought. Though leads to action, action leads to achievement and achievement leads to fulfilment. In DBMS, we get ER models (thought), translate to concrete DB schemas (action) and by performing transactions, we get fulfilment.

00X

Thomas's Write Rule is an improvement to the Basic Timestamp Ordering protocol. Using it enables good concurrency with view serializable schedules. It improves the basic time ordering protocol by rejecting fewer WriteOperations by modifying check Operations for W_item(X). Outdated writes are ignored in Thomas's Write Rule but a Transaction following Basic Time Ordering protocol will abort such a Transaction.

1. If R_TS(X) > TS(T), abort and rollback T and reject the operation.
2. If W_TS(X) > TS(T), don't execute the Write Operation and continue processing.
3. If neither the condition occurs, only then execute the W_item(X) operation of T and set W_TS(X) to TS(T)

00X1

Granularity is the size of data items chosen as the **unit of protection** by a concurrency control protocol. It can be an entire database, a file, a page, a record or a value of a record.

QUERIES

User(*userId*, firstname, lastname, email);
CreditCard (*userId*, *cardNumber*, securityCode, expirationDate);
Item (*itemId*, name, description, pricePerUnit);
Orders (*orderId*, *userId*, *cardNumber*, orderTotalAmount)
ItemsInOrder (*orderId*, *itemId*, quantity);

- Create Table Orders,
- Items that didn't sell
- Emails of users who spent $100 or more per credit card (must not be one order)
- Delete all users which have all cards expired. If a user has at least 1 non-expired cards, don't delete

```
CREATE TABLE Orders (
       orderId VARCHAR(30) NOT NULL PRIMARY KEY,
       userId VARCHAR(30) NOT NULL,
       cardNumber VARCHAR(30) NOT NULL,
       orderTotalAmount MONEY NOT NULL,
       FOREIGN KEY (userId) REFERENCES Users (userId)
              ON DELETE NO ACTION
              ON UPDATE CASCADE,
       FOREIGN KEY (cardNumber) REFERENCES CreditCard (cardNumber)
              ON DELETE NO ACTION
              ON UPDATE CASCADE
);

SELECT *
FROM Item
WHERE itemId NOT IN (
       SELECT itemId
       FROM ItemInOrder
)

SELECT DISTINCT email
FROM (
       SELECT u.email, c.cardNumber, SUM(o.orderTotalAmount) as total
       FROM Users u JOIN CreditCard c ON u.userId = c.userId
       JOIN Orders o  ON o.cardNumber = c.cardNumber
       GROUP BY u.email, c.cardNumber
       HAVING SUM(o.orderTotalAmount) >= 100
) as temp
```

```sql
DELETE
FROM Users
WHERE userId NOT IN (
       SELECT u.userId
       FROM Users u JOIN CreditCard c ON u.userId = c.userId
       WHERE c.expirationDate > GETDATE()
)
AND userId IN (
       SELECT userId FROM CreditCard
)
```