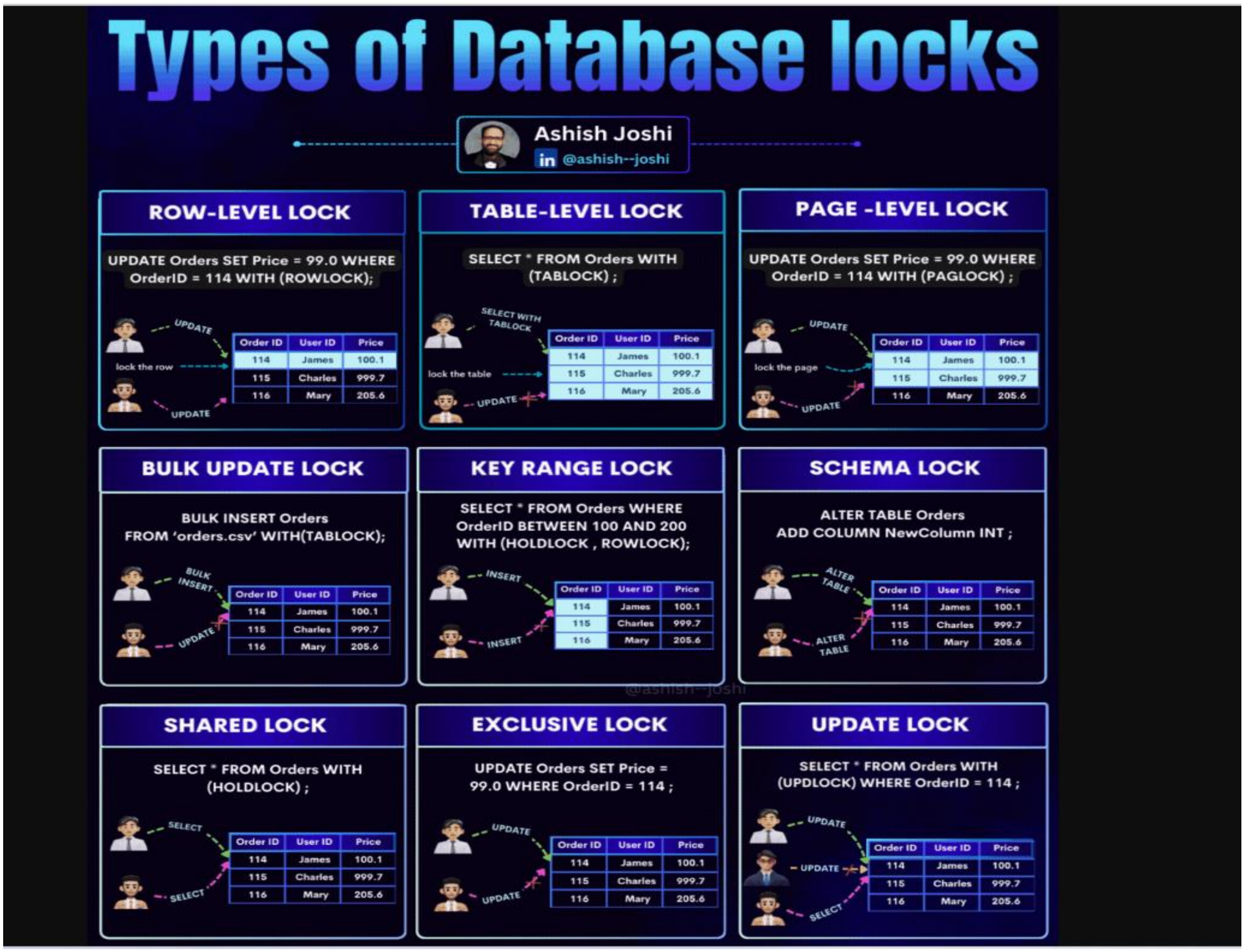

Types of Database locks

ROW-LEVEL LOCK

```
UPDATE Orders SET Price = 99.0 WHERE
OrderID = 114 WITH (ROWLOCK);
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

TABLE-LEVEL LOCK

```
SELECT * FROM Orders WITH
(TABLOCK) ;
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

PAGE -LEVEL LOCK

```
UPDATE Orders SET Price = 99.0 WHERE
OrderID = 114 WITH (PAGLOCK) ;
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

BULK UPDATE LOCK

```
BULK INSERT Orders
FROM 'orders.csv' WITH(TABLOCK);
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

KEY RANGE LOCK

```
SELECT * FROM Orders WHERE
OrderID BETWEEN 100 AND 200
WITH (HOLDLOCK , ROWLOCK);
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

SCHEMA LOCK

```
ALTER TABLE Orders
ADD COLUMN NewColumn INT ;
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

SHARED LOCK

```
SELECT * FROM Orders WITH
(HOLDLOCK) ;
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

EXCLUSIVE LOCK

```
UPDATE Orders SET Price =
99.0 WHERE OrderID = 114 ;
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

UPDATE LOCK

```
SELECT * FROM Orders WITH
(UPDLOCK) WHERE OrderID = 114 ;
```

Order ID	User ID	Price
114	James	100.1
115	Charles	999.7
116	Mary	205.6

❖ In Postgres, locks happen at two levels:

rows and tables.Both matter for concurrency, but they behave differently.

❖ Row-level locks
- UPDATE and DELETE automatically take exclusive row locks (block other writers on the same row).
- SELECT ... FOR UPDATE does the same — requested explicitly when you want to read then update.
- SELECT ... FOR SHARE takes a shared row lock (many readers can share, writers wait).

- Plain SELECT takes no row locks at all — it uses MVCC snapshots, so readers don't block writers.

❖ Table-level locks
Every SQL command also takes a table lock to protect existence and schema. Examples:
- SELECT → ACCESS SHARE (blocks schema rewrites like DROP/TRUNCATE (requiring ACCESS EXCLUSIVE), but not DML).
- INSERT/UPDATE/DELETE → ROW EXCLUSIVE (blocks some schema changes, but allows normal queries).
- DROP TABLE, TRUNCATE, VACUUM FULL → ACCESS EXCLUSIVE (blocks everything).

❖ Quick cheat sheet
- SELECT → ACCESS SHARE, no row locks.
- INSERT → ROW EXCLUSIVE + none (except uniqueness checks briefly lock matching rows)
- UPDATE/DELETE → ROW EXCLUSIVE + row locks.
- SELECT ... FOR UPDATE → ACCESS SHARE + row locks.

❖ Why this matters
- Normal reads never block each other because of MVCC.
- But every statement still locks the table for schema safety.
- Schema changes (ALTER, DROP, CREATE INDEX) take stronger locks — on large tables this can look like an outage.

Database Lock Types & Differences

Lock Type	Purpose	Access Allowed	Concurrency Impact
Shared Lock (S)	Allows concurrent reads	Multiple transactions can read but **no modifications**	**High** – many readers allowed
Exclusive Lock (X)	Grants full control for read/write	Only one transaction; blocks all others (read/write)	**Low** – most restrictive
Update Lock (U)	Prevents deadlocks during updates	Data can be read by others, but no exclusive locks allowed	**Medium** – balances concurrency & safety
Schema Lock (Sch-S / Sch-M)	Protects schema objects during changes	Blocks schema modifications depending on type (Sch-S: stability, Sch-M: modification)	**Varies** – stability lock allows queries, modification lock blocks all
Bulk Update Lock (BU)	Optimizes bulk inserts/updates	Reduces lock overhead for bulk operations	**Medium–Low** – less concurrency, more performance

Lock Type	Purpose	Access Allowed	Concurrency Impact
Key-Range Lock	Prevents phantom reads (range queries)	Locks a range of index keys, prevents new inserts in range	**Medium** – consistency ensured for range queries
Row-Level Lock	Locks a single row only	Other rows remain fully accessible	**Very High** – best concurrency inside a table
Page-Level Lock	Locks a data page (several rows)	All rows in that page are locked	**Moderate** – more restrictive than row, less than table
Table-Level Lock	Locks the entire table	No other transaction can access the table	**Very Low** – simplest but most restrictive

Database locks are essential for maintaining data integrity and consistency in a multi-user environment. Each type of lock serves a specific purpose and impacts the way transactions can interact with the database.

1. **Shared Lock (S)**
- Purpose: Allows multiple transactions to read the same data concurrently.
- Access: Only read access is permitted; no modifications are allowed.
- Concurrency: High, as multiple transactions can hold a shared lock on the same resource.

2.**Exclusive Lock (X)**
- Purpose: Grants a transaction exclusive access to read and modify data.
- Access: Only one transaction can hold an exclusive lock, preventing others from reading or writing to the same resource.
- Concurrency: Low, as it restricts all other access to the resource.

3. **Update Lock (U)**
- Purpose: Prevents deadlocks during updates by indicating intent to modify data.
- Access: It allows other transactions to read the data but prevents them from acquiring exclusive locks.
- Concurrency: Balances concurrency with the need to avoid deadlocks.

4**Schema Lock (Sch-S / Sch-M)**
- Purpose: Protects the structure of database objects like tables or indexes during schema changes.
- Access: Prevents changes to the schema by other transactions.
- Concurrency: Depends on the type of schema lock (e.g., schema modification lock (Sch-M) vs. schema stability lock (Sch-S)).

5. **Bulk Update Lock (BU)**
- Purpose: Optimizes performance during bulk insert operations by minimizing the number of locks required.
- Access: Allows bulk operations while reducing locking overhead.
- Concurrency: May reduce concurrency during the bulk operation but improves performance.

6. Key-Range Lock
- Purpose: Prevents phantom reads by locking a range of indexed data.

- Access: Protects against the insertion of new rows in a specific range during a transaction.
- Concurrency: Ensures consistency in range queries while allowing other transactions to access different ranges.

7.**Row-Level Lock**
- Purpose: Locks a single row within a table, allowing other rows to be accessed concurrently.
- Access: Only the specific row is locked, permitting high concurrency within the table.
- Concurrency: Very high, as other rows remain accessible.

8.**Page-Level Lock**
- Purpose: Locks a specific page (block of data) in the database, which contains multiple rows.
- Access: Affects all rows within the locked page.
- Concurrency: Moderate, as it locks multiple rows at once, impacting concurrency more than row-level locks but less than table-level locks.

9.**Table-Level Lock**
- Purpose: Locks the entire table, restricting all access to it.
- Concurrency: Very low, as it blocks all other access to the table, making it the simplest but most restrictive lock.

Here are the common types of locks used in databases:

1. Shared Lock (S Lock)
It allows multiple transactions to read a resource simultaneously but not modify it. Other transactions can also acquire a shared lock on the same resource.

2. Exclusive Lock (X Lock)
It allows a transaction to both read and modify a resource. No other transaction can acquire any type of lock on the same resource while an exclusive lock is held.

3. Update Lock (U Lock)
It is used to prevent a deadlock scenario when a transaction intends to update a resource.

4. Schema Lock
It is used to protect the structure of database objects.

5. Bulk Update Lock (BU Lock)
It is used during bulk insert operations to improve performance by reducing the number of locks required.

6. Key-Range Lock
It is used in indexed data to prevent phantom reads (inserting new rows into a range that a transaction has already read).

7. Row-Level Lock
It locks a specific row in a table, allowing other rows to be accessed concurrently.

8. Page-Level Lock

It locks a specific page (a fixed-size block of data) in the database.

9. Table-Level Lock
It locks an entire table. This is simple to implement but can reduce concurrency significantly.

MVCC (Multi-Version Concurrency Control) is a database management technique used to handle simultaneous transactions without locking the database.
It creates a new version of a data item whenever it is updated, allowing multiple transactions to access the database concurrently.
Each transaction sees a snapshot of the database at a particular point in time, which improves performance and reduces conflicts.
This method ensures consistency and isolation, allowing transactions to read old versions of data while new updates are being made,
thus supporting high levels of concurrency and providing more efficient database operations.