COMP23111 Database Systems

MySQL keywords and their uses include, but not limited to the following:

- **SELECT** column(s); **WHERE** bool exp.
- bool exp. **AND** bool exp
- bool exp. **OR** bool exp
- **NOT** bool exp.
- **ORDER BY** column(s) (**ASC|DESC**)
- **INSERT INTO** table (column(s)) **VALUES** (field(s))
- column **IS NULL** [bool exp.]
- column **IS NOT NULL** [bool exp.]
- **UPDATE** table **SET** column = value (s) (WHERE)
- **DELETE FROM** table/table alias (es) WHERE bool exp.
- … **LIMIT** number
- column **LIKE** '%_pattern' [bool exp.]
- column **IN** (value(s)) [bool exp.]
- column **BETWEEN** value1 AND value2 [bool exp.]
- column/table **AS** alias
- table left **INNER/LEFT/RIGHT/CROSS JOIN** table right **ON** bool exp.
- query **UNION** (ALL) query
- **GROUP BY** column(s)
- **HAVING** bool exp.
- **EXISTS** (query) [bool exp.]
- column op. **ANY/ALL** (query) [bool exp.]
- **CASE WHEN** bool exp. **THEN** result **ELSE** result END
- **IF** bool exp. **THEN** query **ELSEIF** bool exp. **THEN** query **ELSE** query **END IF**;
- **CREATE INDEX** ON table (column(s))
- **CREATE VIEW** name AS query
- **CREATE TRIGGER** name BEFORE/AFTER INSERT/UPDATE/DELETE ON table BEGIN query END
- **DELIMITER** $$ **CREATE PROCEDURE** name (IN(s), OUT(s), INOUT(s)) **BEGIN** query **END**$$ **DELIMITER** ;
- **CALL** procedure()
- **DECLARE** name TYPE (**DEFAULT** value)
- **SET @**variable = value
- Nested query: (query)

MySQL Functions include but not limited to the following:
MIN(column); **MAX**(column); **COUNT**(column); **AVG**(column); **SUM**(column); **ABS**(value); **CEIL**(num); **FLOOR**(num); **SIGN**(num); **ROUND**(num, digits); **IFNULL**(column/value, result); **CURRENT_DATE**(); **CURRENT_TIME**(); **CURRENT_TIMESTAMP**(); **YEAR**(date); **MONTH**(date); **MONTHNAME**(date);

NoSQL Data Modelling principles and approaches

Type	Model	Condition	Model
One-to-One	Nested Object	Read parent	Separate Documents
One-to-Many	Nested Object	Read parent and child	Nested Object
Many-to-One	Separate Documents	Write parent *or* child	Separate Documents
Many-to-Many	Separate Documents	Write parent *and* child	Nested Object

Normalisation helps prevent anomalies – including insertion (omission due to lack of other data), deletion (unintended loss due to deletion of other data) and update (data inconsistency due to redundant data and partial updates).
1NF requires there exist no ***repeating groups***, and values in each column are *atomic*, or single-valued.
2NF *additionally* requires no ***partial dependencies*** – all non-key attribute is *functionally dependent* on PK.
3NF *additionally* requires no ***transitive dependencies*** – all attributes cannot be computed though another.

Weak Entity is a type of entity that cannot be uniquely identified by its attributes alone; therefore, it must use a *foreign key* (FK) in conjunction with its attributes to create a *primary key* (PK). The FK is typically a PK of an entity it is related to.

Design Phrases of Database Application proceeds as following:
Conceptual Design / Data Model -> Logical Design / Data Model -> Physical Design / Data Model -> Internal Schema -> Transaction / Application Design.

ACID (Atomicity, Consistency, Isolation, Durability) of a database transaction:
Atomicity: Either all occur, or nothing occurs.
Consistency: Leave the database in a consistent state.
Isolation: Does not affect the execution of other transactions.
Durability: Effects must be permanent, even in the event of a system failure.

Shorthand Notation for describing a schedule. They collectively appear as $S\ is\ a\ series\ of\ actions$).

- **b(T)**: Indicates the beginning of a transaction T.
- **r(T, x)**: Indicates that transaction T is reading data item x.
- **w(T, x)**: Indicates that transaction T is writing to data item x.
- **e(T)**: Indicates the end of transaction T.
- **c(T)**: Indicates that transaction T is committing, permanently applying its updates to the database.
- **a(T)**: Indicates that transaction T terminates now.

CAP Guarantees by Eric Brewer for a distributed data store includes ***C**onsistency* (same everywhere), ***A**vailability* (always online) and ***P**artition Tolerance* (continues working even if some messages are dropped or delayed). It is *impossible* to achieve all three.

Conflicting Operations: (conditions can be swapped by more *exclusive* statements)

1. *At least* one of the operations is trying to write.
2. Conflicting operations belongs to *different* transactions.

Incorrect Summary: Transaction B is doing calculations and read some original and some modified values as Transaction A applies changes slightly before Transaction B reads, yielding an invalid result.
Temporary Update: Transaction B read from a data that has been written by Transaction A. In case Transaction A failed, the value read by Transaction B is not valid and has read a temp update.
Unrepeatable Read: Transaction B read a value before Transaction A updates it later in the sequence. After update, the value Transaction B got will be unrepeatable as later read will get a different one.
Phantom Read: Transaction A deleted the value Transaction B just read. Any further attempt to read the same value will yield an error. Transaction preformed a phantom read.
Lost Update: Transaction B's update to a value got overwritten by Transaction A in the same sequence as the write action happens later. The update by Transaction B is lost.