Data management: used to capture, store, retrieve, analyze, present and interpret (large amount of) data

Database: an organized collection of related data, usually stored on disk [Data abstraction: abstract view of data]

Database Management System (DBMS): a software system designed to store and manage databases

Entity-relationship (ER) diagram: a graph represents the main entities and corresponding relationships

- **Entity**: an object/ entity having a set of attributes (Entity set: the collection of all entities)
- **Attributes**: each attribute has a domain & each entity set has a key attribute
- **Relationship** (relationship degree: the number of participating entity types e.g. Binary R', Ternary R', N-ary R')
- **Weak entities**: identified uniquely only by considering PK of another (owner) entity
- **Specialization** (subclass – has own specific attributes/R'ship) **& generalization** (superclass) [Extended ER (EER)]

Data model: collection of concepts that can be used to describe structure of database to achieve data abstraction

Schemas: metadata or data describing data – static & **Instance**: data in the database at particular time – dynamic

Relational model

Relations (R): a set of records (like table BUT not have merge columns/rows) Domains (D): a set of atomic values

Attributes (A): name of role played by some domain in relation (degree of R: number of attributes in relation)

Tuples (T): ordered list of n values $t =< v_1, v_2, \dots, v_n >$ (relations are sets of tuples)

Three-schema architecture

- External level – provides access to particular parts of the database to users
- Conceptual/logical level – structure of a whole database for a community of users
- Internal level – the physical storage structure of the database

Data independence: ability to change the schema at one level of a database system without having to change the schema at the next higher level

Integrity constraints (ICs): conditions that must true for any instance in DB (DBMS shouldn't allow illegal instances)

- ICs are specified when schema is defined - ICs are checked when relations are modified

Types of ICs: [sequence to check violation of ICs: primary key (Entity, Key), Domain, Referential]

- **Domain constraints**: each attribute in a relation must belong to some domain (right stuff is in the right spot)
- **Key constraints (uniqueness constraint)**: $(\text{Candiate key} + \text{Primary key}) = \text{Key (minimal superkey)} \subseteq \text{Superkey}$
- **Entity constraints**: no PK can be null
- **Referential integrity constraints** (data exists in 'parent' before 'child')
- **User-defined constraints**: general user defined constraints (implemented by: checks, assertions and triggers)

Mapping ER diagrams to relational models 【remember to add foreign key/ reference】

Conceptual perspective design: the Entity Relationship (ER) & Logical design (mapping) – the relational model

1. Strong entity – include all simple attributes of itself & Choose one key attribute as primary key (PK)
2. Weak entity – all simple attributes of itself & PK = PK of parent +PK of weak)
3. Binary 1:1 – choose total participation side, add the PK of another side as FK
4. Binary 1:N – choose N-side, add the PK of another side as FK (only 2 don't need to create a new relations/ table)
5. Binary M:N – include any simple attributes itself & PK of both side as FKs & PK of itself = the combination of FKs

[Notice: 1:1, 1:N can map in the same way as M:N → advantage: reduces 'null' that appear as FK values]

6. Multi-valued attribute – PK = multi-value attribute (simple components) and PK of the entity (no non-key attribute)

【V.S. weak entity: weak entity have other non-key attribute】

7. N-ary R' – PK of all sides as FKs & any simple attributes of n-ary R' & combination of FKs (with N side) as PK itself
8. Super & subclasses (mapping of EER) – create a relational table for the superclass & each subclass

The PK of each subclass is PK of the superclass (it also is FK)

Decomposing a relation (1 → many tables) – replace the relation R by 2 or more relations such that:

Each new relation contains a subset of the attributes of R & Every attribute of R appears in at least one new relation

Join: $R_1 \bowtie R_2$ – (natural) join of 2 relations – each R_1's tuple links with every R_2's tuple having the same values on the common attributes

Lossless-join decompositions: $R = R_1 \bowtie R_2$ (break a relation, and we get exactly the same R back together with join)

Lossy-join decomposition (loss – loss of information OR addition of spurious information)

Commented [XC1]: Usages: Maintain integrity constraints & Control redundancy & Create, modify, and query a database & Provide support for decision making
Control access & Permit concurrent access (multiusers) & Provide loading, backup, and recovery

Commented [XC2]: Important data & Shared & Secured & Well-designed (minimal redundancy) & Variable size

Commented [XC3]: Relationship constraints: limit the possible combination of entities that may participate in the relationship set
- Cardinality constraint: one-to-one, one-to-many, many-to-many
- Participation constraints: total (all participate in relationship) OR partial participation

Commented [XC4]: 1-to-many relationship (1-owner, N-weak entities)
Weak entity has total participation

Commented [XC5]: Logical data independence: ability to change conceptual schema without changing applications
Physical data independence: ability to modify physical schema without changing logical schema

Commented [XC6]: Superkey (SK): subset of attributes that is uniqueness of a relation schema (can have redundant attributes)
Key (K): a minimal Superkey (Candidate key & PK – only one is selected as PK)

Commented [XC7]: Special situation: self referencing relations
Foreign keys: relate 2 schemas – $t_1[FK] = t_2[PK]$ or $t_1[FK]$ is null (R_1 FK have same domain as R_2 PK)
[Deletion: 'child' → directly delete & 'parent' → check referential integrity Modify: opposite of deletion]

Commented [XC8]: Reverse engineering (relational model → ER diagram) - Same order as mapping
Weak entities V.S. Multi-valued attributes – Multi-valued attributes NO non-key attributes
Modification anomaly: data inconsistency that results from data redundancy
Deletion anomaly: loss of certain attributes because of the deletion of other attributes
Insertion anomaly: cannot insert attributes without presence of other attributes (key cannot be null – entity constraint)
Update anomaly: Incorrect grouping → inconsistent data or even loss of data

Functional dependencies (FD) – one attribute determines another attribute

key: minimal set of attributes functionally determines all attributes

Closure of F: the set of all FDs implied by F (closure for a set of attributes X is denoted by $X^+ = \{X, \dots\}$)

Armstrong's Axioms (X, Y, Z are sets of attributes): - Reflexivity: if $Y \subseteq X$, then $X \to Y$

- Transitivity: if $X \to Y$ and $Y \to Z$, then $X \to Z$ - Augmentation: if $X \to Y$, then $XZ \to YZ$ for any Z

Additional rules: - Union: if $X \to Y$ and $X \to Z$, then $X \to YZ$ - Decomposition: if $X \to YZ$, then $X \to Y$ and $X \to Z$

Role of FDs in detecting redundancy (suppose 3 attributes, A, B, C):

- No FDs hold: there is no redundancy here
- Given A → B: several tuples could have the same A value, and if so, they'll all have the same B value

Normalization: process of removing redundancy from data – achieve better designed schemas using FDs and PKs

- **1NF** – only have single value as attribute (BUT have redundancy OR too many NULL value)
- **2NF** – FD: X→Y, where X is minimal key and Y is non-prime attribute, then no proper subset of X determines Y
- **Boyce-Codd (BCNF)** – for every FD: if X → b (non-trivial: b is not the subset of X), then X is a Superkey for R
  【LHS is Superkey, RHS is not part of LHS】 √: no redundancy&anomalies ×: cannot preserve all dependencies
- **Third normal form (3NF)** – (RHS is not part of LHS) LHS is Superkey OR RHS is prime attribute
  √: keep some redundancy/anomalies x: preserve all FDs
  Minimal cover (G) of set F: RHS is single attribute & cannot delete any FD or attribute

Denormalization – violate normal form to gain performance improvements: fewer joins & reduces number of FKs

SQL (structured query language) – declarative, based on relational algebra【result of SQL query is table/relation】

Data definition language (DDL) – define database schema/ structure – e.g. CREATE, ALTER and DROP TABLE

CREATE TABLE: create a new relation (specifying table name, attributes and constraints)

Domain constraint specified for each attribute – can specified directly or by CREATE DOMAIN

Enforcing referential IC → referential triggered actions when referenced tuple is deleted or updated:

- NO ACTION (default): delete/update is rejected
- SET DEFAULT
- SET NULL
- CASCADE: delete/update all roles that refer to

ALTER TABLE: alter the structure of tables
(add/ drop column OR change column definition/domain OR add/ drop constraints)

DROP TABLE: delete tables (includes all tuples, ICs, FKs in other table) from DB

```
ALTER TABLE <table name>
  ADD <column name> <column type>
  [<attribute constraint>] {, <column name>
  <column type> [<attribute constraint>] }
  | DROP <column name> [CASCADE]
  | ALTER <column name> <column-options>
  | ADD <constraint name> <constraint-options>
  | DROP <constraint name> [CASCADE];
```

Data manipulation language (DML) – manipulate/ managing data

SELECT: retrieve data from a database; **INSERT**: insert data into a table; **UPDATE**: updates existing data within a table; **DELETE**: deletes records from a table

```
DROP TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

```
INSERT INTO Student (sID, sName, GPA, sizeHS)
VALUES (53688, 'Smith', 3.2, 200)
```

```
UPDATE <table name>
  SET <column name> = <value expression>
      {, <column name> = <value expression>}
  [WHERE <select condition>];
```

Relational algebra:

- **Projection (SELECT)**: vertically select attributes wanted to return
  Can evaluate expressions (+, -, *, /) (e.g. SELECT Year+2)
  Rename attributes: *old-name* AS *new-name* (e.g. SELECT Role AS Role1)
- **Selection (WHERE: join condition & search condition)**: horizontal scanner (select tuples)
- **Join (FROM & WHERE join condition)**
  – join 2 tables: R1, R2 on their shared attribute
  Based on Cartesian product (give all the combination) – n rows in R_1, m rows in R_2 → R with $m * n$ rows
  Types of join operations (3) in WHERE clause:
  Theta-join: with $(=, \neq, <, \leq, >, \geq)$
  Equi-join: with $=$ (special case of Theta join)
  Natural join (Equi-join): columns with same name of associate tables will appear once only [if having no pairs of common attribute → return cross product of 2 tables]
  Inner join (default) – includes the tuple only if matching tuples exist in both relations
  Outer join – include tuples that do not satisfy the join condition [set NULL value to tuple that does not match]
- **Sorting (ORDER clause)**: order result tuples according to given sort key [ASC (default): small to large/ DESC]

```
DELETE FROM <table name>
  [WHERE <select condition>];
```

```
SELECT <attribute list>
FROM table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference ON
< search_condition>
```

Set operations – relation is set of tuples (no duplicates) 【must be **union compatible** to do operations: iff 2 relations have same degree (number of columns) & each column has same domains】

Commented [XC9]: Decomposing into BCNF losslessly (3 steps):
1. Add implicit FDs to your list of FDs
2. Pick FDs that violates BCNF of the form $X \to b$
3. Decompose R: $R_1(X \cup \text{others})$ & $R_2(X \cup b)$ [Note: table with only 2 attributes is in BCNF]

Dependency preservation: FD $X \to Y$ is preserved, having all attributes of FD in one table → hard to check violation of FD without join back into a single table

Commented [XC10]: Decomposition into 3NF (using minimal cover):
1.Get the minimal cover F' & find all keys
2.Decompose using F' if violating 3NF (same step as BCNF include decompose the implicit FDs)
3.Identify not preserved FDs in F' → add back

Commented [XC11]: Finding minimal covers of FDs (do in order):
1. Put FDs in standard form (only 1 on RHS)
2. Minimize LHS of each FD
3. Delete redundant FDs (closures are the same with and without the redundant FD)

Commented [XC12]: CREATE TABLE StarsIn(
StarID - - INTEGER, ··· ···,
PRIMARY KEY (StarID, MovieID),
FOREIGN KEY (MovieID) REFERENCES (Movies),
ON DELETE(UPDATE) CASCADE)

Commented [XC13]: Varchar (variable character) V.S. character (fixed length character)

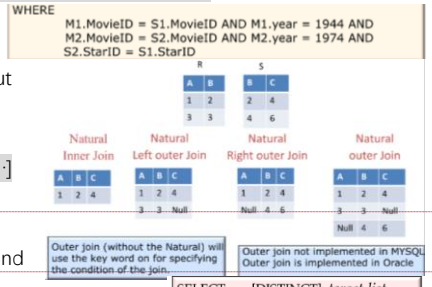
Commented [XC14]: CASCADE: drop things related to that table
RESTRICT: disallow dropping the table, if that table got somethings related to it

Commented [XC15]: ·Substring comparisons (LIKE, IN, IS)
%: 0 or more arbitrary characters
_ (underscore): any one character
·Arithmetic operators and functions
+, -, *, /, date, time, year, BETWEEN ··· AND ···, etc (e.g. WHERE Year(Sys_Date – Bdate) > 55)

Commented [XC16]: Full outer join – includes all rows from both tables
Left/right outer join – includes all rows on left/ right table

Commented [XC17]: retain duplicates: UNION ALL, INTERSECT ALL, EXCEPT ALL [relation r, s have m, n duplicates, then number of duplicates will be: UNION ALL: $m + n$ & r INTERSECT ALL s: $\min(m, n)$ & r EXCEPT ALL s: $\max(0, m - n)$]

- **UNION (∪)**: SELECT ··· UNION [ALL] SELECT ··· [UNION [ALL] SELECT ···] same as OR using in WHERE clause
- **INTERSECTION (∩)**: SELECT ··· INTERSECT [ALL] SELECT ··· [INTERSECT [ALL] SELECT ···]
  【Note: INTERSECT is part of the SQL standard, BUT is not implemented in MySQL → do it in 2 table】
- **DIFFERENCE/EXCEPT/MINUS (−)**: includes all tuples in R_1, but not in R_2 【Note: EXCEPT can use in SQL, BUT cannot use in MySQL → nested queries can work as EXCEPT】
  SELECT ··· EXCEPT [ALL] SELECT ··· [EXCEPT [ALL] SELECT ···]



Aggregation: produce summary values on SELECT clause (can't nested aggregation)

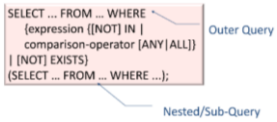
GROUP BY and HAVING – divide tuples into groups (GROUP BY) and apply conditions to each group (HAVING)

```
SELECT    [DISTINCT] target-list
FROM      relation-list
WHERE     qualification
GROUP BY  grouping-list
HAVING    group-qualification
ORDER BY  target-list
```

Conceptual evaluation of a query:
1. Compute cross-product of relation-list (FROM); 2. Keep tuples that satisfy qualification (WHERE); 3. Groups remaining tuples in grouping-list (GROUP BY); 4. Keep groups that satisfy group-qualification (expressions in group-qualification have 1 value per group) (HAVING); 5. Delete fields that are not in target-list (SELECT) → Generate 1 answer tuple per qualifying group; 6. DISTINCT, eliminate duplicate rows; 7. ORDER BY, sort the results

Nested queries/sub-query: query within query – useful where data is fetched and used in comparison condition

· Inside WHERE of another SELECT statement · Inside an INSERT, UPDATE or DELETE statement · Sub-query cannot include ORDER BY, but DISTINCT may order results of a sub-query



- **Correlated and non-correlated variants (nested queries)**
  Non-correlated – inner, outer query run independently (don't need to do join) → computed just once (evaluated from the 'inside out') – e.g. IN, NOT IN & Correlated – inner depends on outer query (need to do join in inner query WHERE) → compute many times – e.g. EXISTS/ NOT EXISTS
- **Sub-query operators** (expression and attribute list in sub-query SELECT clause must have same domain)
  · [NOT] IN (sub-query)
  · Comparison-operator (>, <, =, ≥, ≤, <>) [ANY|ALL] (sub-q) [can only use when sub-query return 1 value]
  Equivalence: 'IN' and '= ANY' & 'NOT IN' and '<> ALL' & Non-equivalence: 'NOT IN' and '<> ANY'
  Sub-queries V.S. set operations V.S. joins (can be used to write the same query)
  · Joins – display results from multiple tables · Sub-queries – compare aggregates to other values

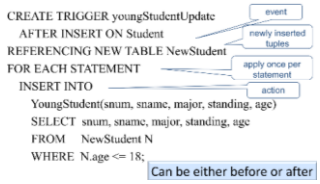
Division – for queries include 'for all'/ 'for every' [BUT no direct way to write division in SQL →double negation]

Views – types: virtual tables (not physically exist) & materialized (physically exist & need to update)

NULL value – indicates value is unknown [IS NULL (IS NOT NULL): used to check whether value is known/unknown]

- Operations on NULL value [NULL requires 3-valued (true, unknown, false) logic using the truth value unknown]
  · OR: (unknown OR true) = true; (unknown OR false) = unknown; (unknown OR unknown) = unknown
  · AND:(true AND unknown) = unknown; (false AND unknown) = false; (unknown AND unknown) = unknown
  · NOT:(NOT unknown) = unknown
  · Comparisons between 2 null values, or between a null and any other value → return unknown [All aggregate operations except COUNT(*) ignore tuples with null values on the aggregated attributes]

Semantic constraints can be specified using CHECK and ASSERTION statements



- **CHECK**– write at the end of CREATE TABLE (check when tuples are inserted or modified) CHECK (condition)
- **ASSERTION** (like CHECK) – check multiple tables CREATE ASSERTION <name> CHECK (NOT EXISTS (SELECT ···))
- **TRIGGER** – procedure that start automatically if specified changes occur in DBMS (may cause cascading effects)

Data warehousing – process of constructing and using data warehouses

Commented [XC18]: SUM/ AVG ([DISTINCT]) expression), COUNT ([DISTINCT] expression), COUNT(*), MAX/ MIN(expression) (Note: can have non-numeric domain)

Commented [XC19]: *Target-list* contains:
1)Attribute names (must also be in *grouping-list*)
Each answer tuple corresponds to a *group*
Group = a set of tuples with same value for all attributes in *grouping-list*
Selected attributes must have 1 value per group
2)Terms with aggregate operations
Group-qualification: attributes are either in *grouping-list* OR an aggregate operator
Having: can also include aggregates

Commented [XC20]: (Check (non)existence of data) T if the result of the correlated sub-query is not-empty/ empty set

Commented [XC21]: ANY: true if one comparison is true [e.g. > ANY X → return the value greater than MIN(X)]
ALL: true if all comparisons are true [e.g. > ALL X → return the value greater than MAX(X)]

Commented [XC22]: A single table that is derived from other tables, which could be base tables or previously defined views
Benefits of using views
Simplification: hide complexity of underlying tables to end-users
Security: hide columns containing sensitive data from certain groups of users
Computed columns: create computed columns, which are computed on the fly
Logical data independence: provide support for logical data independence, that is users and user's programs that access the database are immune from changes in the logical structure of the database (not influence based table)
Defining a view: CREATE VIEW <view name> (<column name> {,<column name>}) AS SELECT ···
Dropping views: DROP VIEW [IF EXISTS] <view name> [,<view name>]··· [RESTRICT | CASCADE]
RESTRICT: drops table, unless there is a view on it
CASCADE: drops table, and recursively drops any view referencing it

Commented [XC23]: Parts of triggers: 1. Event (activates trigger)
2. Condition (tests whether the trigger should run)
3. Action

Different types of needs: - Operational DBMS (for running business) – OLTP - Data warehouse – OLAP

OLAP (on-line analytical processing): technology that enables analysts, managers, and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information

Star schema – one table for fact table (BCNF) + one table per dimension (not normalized/ redundant – fewer joins)

Snowflake schema – one fact table + dimensions (one table per level – normalized)

	OLTP	OLAP
Typical User	Basically Everyone (Many Concurrent Users) general users	Managers, Decision Support Staff (Few)
Type of Data	Current, Operational, Frequent Updates	Historical, Mostly read-only
Type of Query	Short, Often Predictable	Long, Complex
# query	Many concurrent queries	Few queries
Access	Many reads, writes and updates	Mostly reads
DB design	Application oriented	Subject oriented
Schema	E-R model, RDBMS	Star or snowflake schema
Normal Form	Often 3NF	Unnormalized
Typical Size	MB to GB	GB to TB
Protection	Concurrency Control, Crash Recovery	Not really needed (it's backup of data that use to analyse)
Function	Day to day operation	Decision support

Roll-up/ drill-down – like zoom in/out; Slicing – look into 1 value in one of the dimensions but not zoom in/out (use '=')

Dicing – look into one of the values in multiple dimensions

Pivoting – rotate cube to provide an alternative presentation (changing GROUP BY items)

WITH ROLLUP/CUBE (end of GROUP BY) – combine fact data and dimensions [Roll-up, drill-down, slicing, dicing, and pivoting operations are expensive → CUBE pre-calculated]

Number of tuples: (Just GROUP BY all dimension: $a * b * c$)

ROLLUP: $abc + ab + a + 1$ [GROUP BY A, B, C WITH ROLLUP (order matters), can do by UNION]

CUBE: $(a + 1)(b + 1)(c + 1)$ [2^k for computing all cells for a k-dimensional cube, can do by pivoting ROLLUP]

Database security

Privacy: ability of individuals to control the terms under which their sensitive data is acquired and used

Security: a required building block for privacy (reasons: legal/policy, ethical, technical), which includes:

- Preventing storage of sensitive data - Ensuring appropriate/ authorized (access of people) use of sensitive data

Database control measures:

1. Access control (3 mechanisms) – create user accounts and passwords (& give different privilege to accounts)
   Discretionary access control: grant privileges to users (2 level): 1. Account level (high level) – CREATE SCHEMA/ TABLE; 2. Relation level – access matrix model: owner can grant (select, write and references) through views
   Grant privilege: [GRANT SELECT ON V TO B;] (SELECT – read access)
   Revoking of privileges: REVOKE command used to cancel a privilege [REVOKE SELECT ON V FROM B;]
   Spread of privileges using the GRANT OPTION: [GRANT SELECT ON V TO B WITH GRANT OPTION;]
   Mandatory access control: classify data and users into security classes (Top Secret, Secret, Confidential, Unclassified)
   · Simple security property (intuitive) – allowed to read $\text{class(Subject)} \geq \text{class(Object)}$
   · Star property – allowed to write, if $\text{class}(S) \leq \text{class}(O)$ (can only write things $\geq$ to its class) → Prevent information from flowing from higher to lower classification 【the row is removed if the key is NULL】
   Role-based access control (not in user-level) – privileges are based on organizational roles (define user types/roles) → users are assigned to appropriate roles (can use with discretionary and mandatory access control)
   √ : flexibility and better support for web-based applications

```
GRANT ROLE full-time TO emp_typ1;
GRANT ROLE intern TO emp_typ2;
```

2. Inference control – information about individuals cannot accessed (only statistical queries are allowed) · Provide minimum threshold on number of tuples · Prohibit sequences of queries that refer to the same population of tuples · Introduce slight noise or inaccuracy in database but maintain the correct statistical data · Partition the database – store records in groups of minimum size (new tables)
3. Flow control – prevents information from flowing to unauthorized users
4. Data encryption – used to protect sensitive transmitted data (protect data while it's in transaction)

Basic DBMS security functions: 1. User accounts: user log in with username and password; 2. Login session: sequence of database operations by user recorded in system log (log store the info on the users); 3. Database audit: reviewing log to examine all accesses and operations applied (see who make a change)

SQL injection methods:

1. SQL manipulation: changes an SQL command in the application (e.g. OR 'x'='x') to attack occurs during login
2. Code injection: add additional SQL statements or commands that are then processed
3. Function call injection: database or operating system function call inserted into vulnerable SQL statement to manipulate data or make a privileged system call

Commented [XC24]: OLTP (on-line transaction processing)
-information systems that facilitate/manage transaction-oriented applications, typically for data entry and retrieval transaction processing (responds immediately to user requests)
Goals of OLTP: availability, speed, concurrency and recoverability

Commented [XC25]: Data warehouse: subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision-making process ('optimize' business)
-Subject-oriented: data gives information about a particular subject (analyze data for decision makers)
-Integrated: integrating multiple data sources
-Time-variant: all data are associated with a particular time period
-Non-volatile: data is stable in a data warehouse (focus on loading of data & accessing of data)
Separate data warehouse?
-High performance for both systems
Warehouse: needs to be tuned for complex queries, multidimensional views, consolidation
-Require different types of queries
OLAP: use many statistical functions, grouping and aggregation (few but complex queries)

Commented [XC26]: Threats to database security:
-Loss of **C**onfidentiality: unauthorized disclosure of confidential information
-Loss of **I**ntegrity: improper modification of information
-Loss of **A**vailability: legitimate user cannot access data objects
- Unauthorized privilege escalation; - Privilege abuse; - Denial of service (service is there, but it won't allow user to access); - Weak authentication (get some information about the user, e.g. can login if having the username but don't have password)
SQL injection – attacker injects a string input through the (often web/ other interface) application which changes or manipulates SQL statement to attacker's advantage

Commented [XC27]: Sensitive data – information that is protected against unwarranted disclosure
Types of sensitive data: - Inherently sensitive; - From a sensitive source; - Declared sensitive; - sensitive attribute/record; - Sensitivity in relation to previously disclosed data (e.g. data in one database is sensitive to data in another database)