SQL Part 2

CS460 Databases for Data Scientists

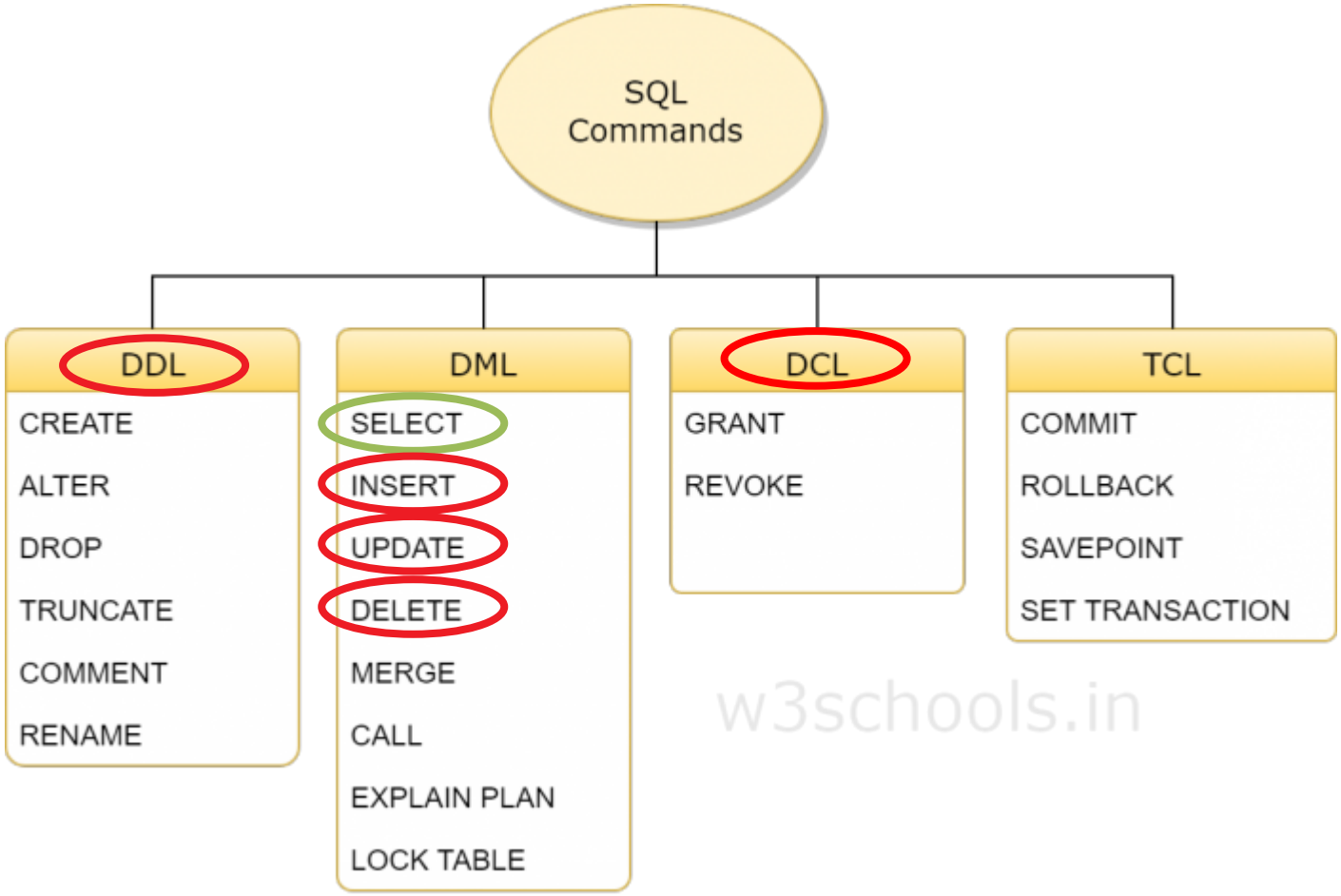
UNIVERSITY OF MANNHEIM
Data and Web Science Group

Outline

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Last week**
  - Overview of The SQL Query Language
  - Basic Query Structure
  - Join Operators
  - Set Operations
  - Aggregate Functions
  - Null Values
  - Subqueries
    - In WHERE part
    - In FROM part
    - In SELECT part
- **Today**
  - Data Definition
  - Data Types in SQL
  - Modifications of the database
  - Views
  - Integrity Constraints
  - Roles & Rights

Parts of SQL: The Big Picture



Source: https://www.w3schools.in/mysql/ddl-dml-dcl/

SQL Data Definition Language (DDL)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Allows the specification of information about relations, including
  - The schema for each relation
  - The domain of values associated with each attribute
  - Integrity constraints
- And as we will see later, also other information such as
  - The set of indices to be maintained for each relations
  - Security and authorization information for each relation
  - The physical storage structure of each relation on disk

Recap: Domain of an Attribute

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- The set of allowed values for an attribute
  - Programmers: think *datatype*

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Simple Domains in SQL

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **CHAR(n)** Fixed length character string, with user-specified length *n.*
- **VARCHAR(n)** Variable length character strings,
  with user-specified maximum length *n.*
- **INT** Integer (a finite subset of the integers that is machine-dependent).
- **SMALLINT** Small integer (a machine-dependent subset of the integer domain type).
- **NUMERIC(p,d)** Fixed point number, with user-specified precision of *p* digits,
  with *d* digits to the right of decimal point.
  (ex., **NUMERIC**(3,1), allows 44.5 to be stores exactly,
  but not 444.5 or 0.32)
- **REAL, DOUBLE PRECISION**
  Floating point and double-precision floating point numbers,
  with machine-dependent precision.
- **FLOAT(n)** Floating point number, with user-specified precision of at least *n* digits.

Date and Time Data Types in SQL

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- We have already encountered characters and numbers
- **DATE**: Dates, containing a (4 digit) year, month and date
  - Example: **DATE** '2005-7-27'
- **TIME**: Time of day, in hours, minutes and seconds.
  - Example: **TIME** '09:00:30' **TIME** '09:00:30.75'
- **TIMESTAMP**: date plus time of day
  - Example: **TIMESTAMP** '2005-7-27 09:00:30.75'
- **INTERVAL**: period of time
  - Example: interval '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

Arithmetics with Dates

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Dates can be compared
  - i.e., < or >
    - e.g., select employees who started before January 1st, 2017
  - Special function: NOW() (in MariaDB; name may differ for other DBMS)
- Dates can be added to / substracted from intervals and other dates
  - e.g., select students who have been enrolled for more than five years
- Implementation often not standardized
  - Details differ from DBMS to DBMS!

User-defined Types

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **CREATE TYPE** construct in SQL creates user-defined type

CREATE TYPE *Dollars* **AS NUMERIC (12,2) FINAL**

required due to SQL standard; not really meaningful

CREATE TABLE *department*(
dept_name **VARCHAR** (20),
building ***VARCHAR*** (15),
budget Dollars
);

User-defined Domains

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **CREATE DOMAIN** construct creates user-defined domain types

  CREATE DOMAIN *person_name* **CHAR**(20) **NOT NULL**

- Types and domains are similar
  - Domains can have constraints, such as **NOT NULL**, specified on them

    CREATE DOMAIN *degree_level* **VARCHAR**(10)
    CONSTRAINT *degree_level_test*
    CHECK (**VALUE IN** ('Bachelors', 'Masters', 'Doctorate'));

Domain Constraints vs. Table Constraints

- Some checks may reoccur over different relations
  - e.g., degrees for students or instructors
  - e.g., salutations
  - e.g., valid ranges for ZIP codes

- Binding them to a *domain* is preferred
  - Central definition
  - Consistent usage

Large Object Types

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:
  - **BLOB**: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - **CLOB**: character large object -- object is a large collection of character data
- When a query returns a large object, a pointer is returned rather than the large object itself

Creating Relations

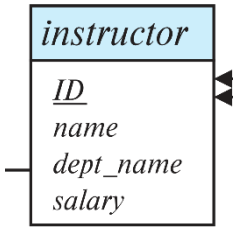
- An SQL relation is defined using the **CREATE TABLE** command:

  CREATE TABLE *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the datatype/domain of values in the domain of attribute A_i

- Example:

  CREATE TABLE *instructor* (
  ID **CHAR**(5),
  name **VARCHAR**(20),
  dept_name **VARCHAR**(20),
  salary **NUMERIC**(8,2))



Recap: Keys

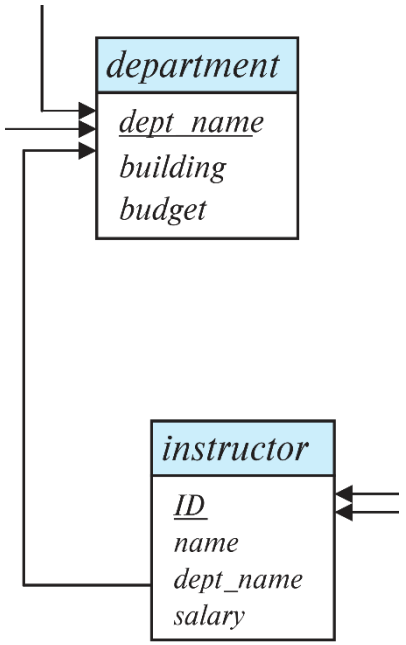
UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Primary keys identify a unique tuple of each possible relation *r(R)*
  - Typical examples: IDs, Social Security Number, car license plate
- Primary keys can consist of multiple attributes
  - e.g.: course ID plus semester (CS 460, FSS 2019)
  - Must be minimal – (ID, semester, instructor) would work as well
- Foreign keys refer to other tables
  - i.e., they appear in other tables as primary keys

Defining Keys

- **PRIMARY KEY** (A_1, ..., A_n)
- **FOREIGN KEY** (A_m, ..., A_n) **REFERENCES** *r*
- *Example:*

CREATE TABLE *instructor* (
ID **CHAR**(5),
name **VARCHAR**(20),
dept_name **VARCHAR**(20),
salary **NUMERIC**(8,2))
PRIMARY KEY (*ID*),
FOREIGN KEY *(dept_name*)
REFERENCES *department(dept_name)*
);

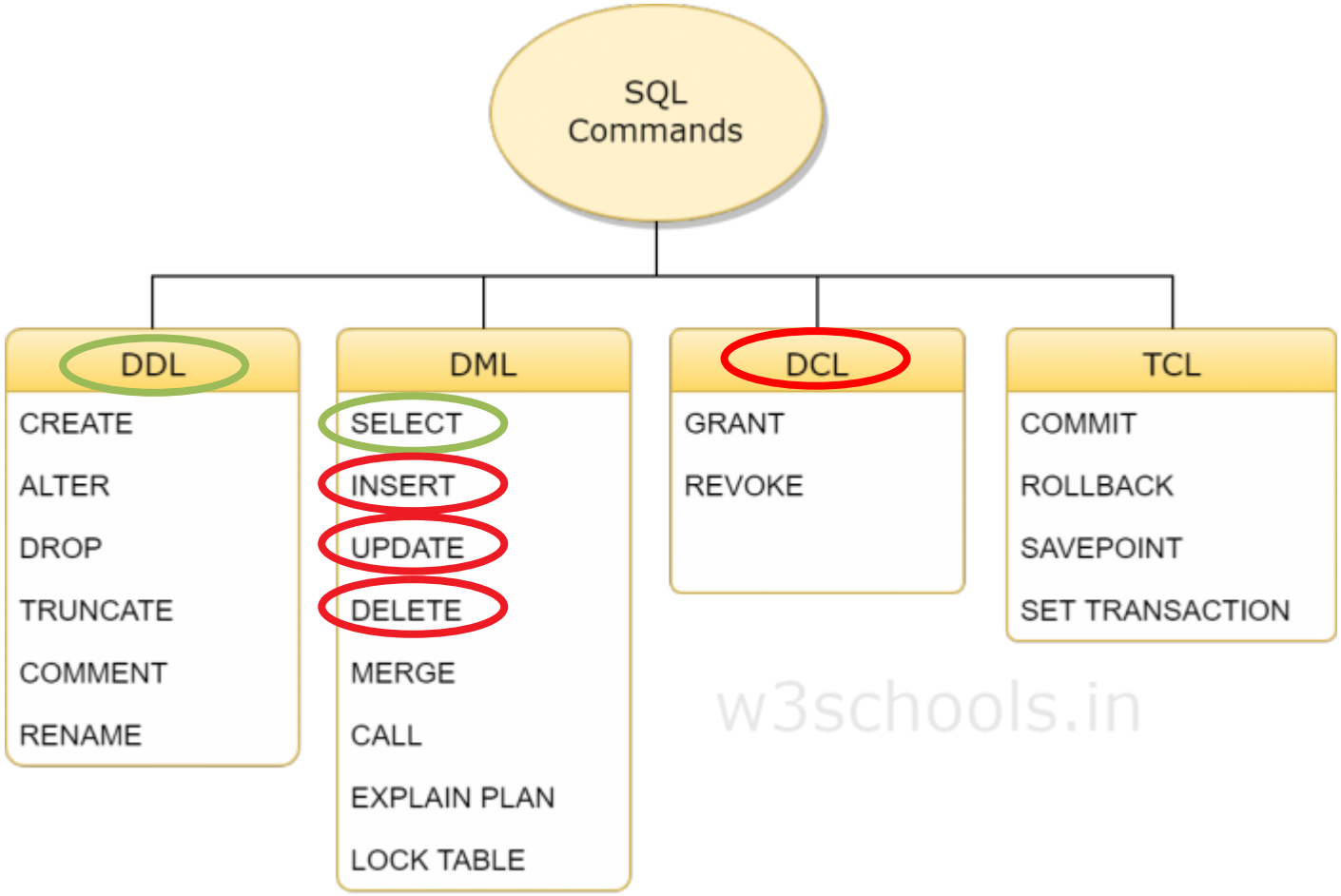


Removing and Altering Relations

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Removing relations
  - **DROP TABLE** *r*
- Altering
  - **ALTER TABLE** *r* **ADD** *A D*
    - where *A* is the name of the attribute to be added to relation *r,* and *D* is the domain of *A*
    - all existing tuples in the relation are assigned ***NULL*** as the value for the new attribute
  - **ALTER TABLE** *r* **DROP** *A*
    - where *A* is the name of an attribute of relation *r*
    - not supported by many databases

Parts of SQL: The Big Picture

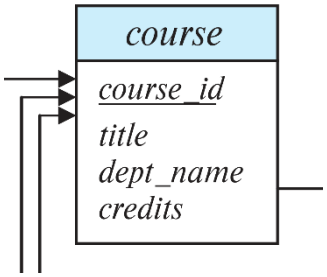


Source: https://www.w3schools.in/mysql/ddl-dml-dcl/

Insertion into a Relation

UNIVERSITY OF MANNHEIM
Data and Web Science Group

Note: there is an inherent ordering in the columns (the one we defined when we entered **CREATE TABLE** ...)



- Add a new tuple to *course*

  INSERT INTO *course*
  VALUES (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- or equivalently

  INSERT INTO *course* (*course_id*, *title*, *dept_name*, *credits*)
  VALUES (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- Add a new tuple to *student* with *tot_creds* set to **NULL**

  INSERT INTO *student*
  VALUES (’3003’, ’Green’, ’Finance’, **NULL**);

Insertion of Data from Other Tables

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Add all instructors to the *student* relation with tot_creds set to 0

  INSERT INTO *student*
  SELECT *ID, name, dept_name, 0*
  FROM *instructor*

- Note: the **SELECT FROM WHERE** statement is evaluated **fully** before any of its results are inserted into the relation

Inserting Data into Relations with Constraints

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Effect of primary key constraints:
  - **INSERT INTO** *instructor* **VALUES** (‘10211’, ’Smith’, ’Biology’, 66000);
  - **INSERT INTO** *instructor* **VALUES** (‘10211’, ’Einstein’, ’Physics’, 95000);
  - ...and we defined ID the primary key!
  - Then the DBMS will return an error

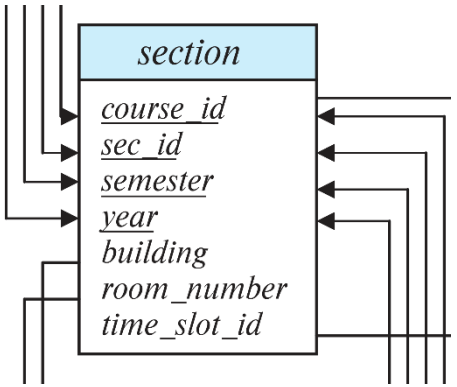
- Effect of **NOT NULL** constraints
  - **INSERT INTO** *instructor* **VALUES** (‘10211’, **NULL**, ’Biology’, 66000);
  - ...and we defined name as NOT NULL!
  - Then the DBMS will return an error

- Recap: DBMS takes care of *data integrity*

Caveats with NOT NULL Constraints

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Rationale:
  - Each course takes place at a specific room and time slot
  - We'll create a **NOT NULL** constraint on those fields



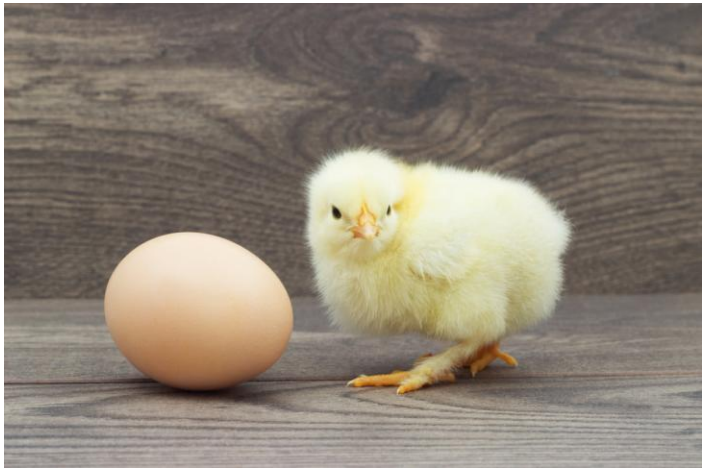
- Use case:
  - First: enter all courses in the system
  - Second: run clever time and room allocation algorithm
    - Which will then fill all the buildings and time slots

Caveats with NOT NULL Constraints (ctd.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Example: every employee needs a substitute
  - **CREATE TABLE** *employee* (
    ID **VARCHAR**(5),
    name **VARCHAR**(20) **NOT NULL**,
    substitute **VARCHAR**(5) **NOT NULL**,
    PRIMARY KEY *(ID),*
    FOREIGN KEY *(substitute*) **REFERENCES** *employee(ID)*
    );

- What do you think?



Updating Data

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Example: update the salary of a single person

  UPDATE *employee*
  SET *salary* = 80000
  WHERE *person_id* = 43743

- Example: update all salaries by 5%

  UPDATE *employee*
  SET *salary* = *salary * 1.05*

- Example: moving all people from a department to a new building

  UPDATE *employee*
  SET *building* = 'Taylor'
  WHERE *dept_name* = 'Biology'

- Anatomy of an **UPDATE** query
  - **SET** defines which updates to carry out
  - **WHERE** defines which records to update (omitted = all records)

Updating Data

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Cut salaries above 100,000 by 5%, below 100,000 by 3%
- Write two **UPDATE** statements:

Thought experiment:
Tom's salary is 102,000

```
UPDATE instructor
   SET salary = salary * 0.95
   WHERE salary > 100000;

UPDATE instructor
   SET salary = salary * 0.97
   WHERE salary <= 100000;
```

- What do you think?
- Should rather be done using the **CASE** statement (next slide)

Conditional Updates with case Statement

- Cut salaries above 100,000 by 5%, below 100,000 by 3%

```
UPDATE instructor
    SET salary = CASE
        WHEN salary > 100000 THEN salary * 0.95
        ELSE salary * 0.97
    END
```

Updates with Subqueries

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Recompute and update tot_creds value for all students

```
UPDATE student S
SET tot_cred = (SELECT SUM(credits)
                FROM takes, course
                WHERE takes.course_id = course.course_id AND
                      S.ID= takes.ID AND takes.grade <> 'F' AND
                      takes.grade IS NOT NULL);
```

  - The above query sets *tot_creds* to **NULL** for students who have not taken or passed any course

- Instead of **SUM**(*credits*), use:

```
CASE
   WHEN SUM(credits) IS NOT NULL THEN SUM(credits)
   ELSE 0
END
```

Deleting from a Relation

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Delete**
  - Remove all tuples from the *student* relation

    DELETE FROM *instructor*
  - May be refined (e.g., only removing *specific* tuples)
    DELETE FROM *instructor* **WHERE …**

Deleting from a Relation

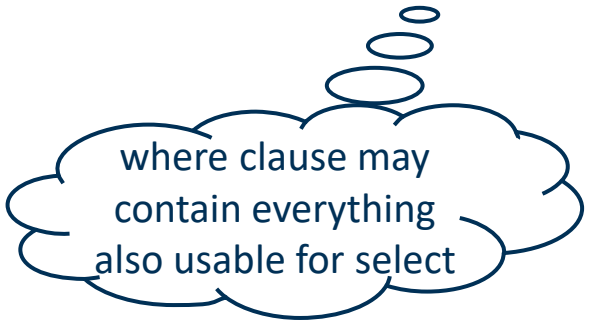
UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Delete all instructors from the Finance department

  DELETE FROM *instructor*
  WHERE *dept_name*= ’Finance’;

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building

  DELETE FROM *instructor*
  WHERE *dept_name* **IN** (**SELECT** *dept_name*
  FROM *department*
  WHERE *building* = ’Watson’);



Deleting from a Relation

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Delete all instructors whose salary is less than the average salary of instructors

  DELETE FROM *instructor*

  WHERE *salary* < (**SELECT AVG** (*salary*)
  FROM *instructor*);

- This would delete five tuples
  - But then, the average changes!

- How does the query behave if the deletion is processed one by one?

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
~~32343~~	~~El Said~~	~~History~~	~~60000~~
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
~~76766~~	~~Crick~~	~~Biology~~	~~72000~~
~~10101~~	~~Srinivasan~~	~~Comp. Sci.~~	~~65000~~
~~58583~~	~~Califieri~~	~~History~~	~~62000~~
83821	Brandt	Comp. Sci.	92000
~~15151~~	~~Mozart~~	~~Music~~	~~40000~~
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Deleting from a Relation

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Delete all instructors whose salary is less than the average salary of instructors

  DELETE FROM *instructor*

  WHERE *salary* < (**SELECT AVG** (*salary*)
  FROM *instructor*);

- Processing this query in SQL
  - First, the **SELECT** query is evaluated
    - i.e., the result is now treated as a constant
  - Then, the **DELETE** statement is executed

DELETE vs. TRUNCATE

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- All records from a table can also be removed using

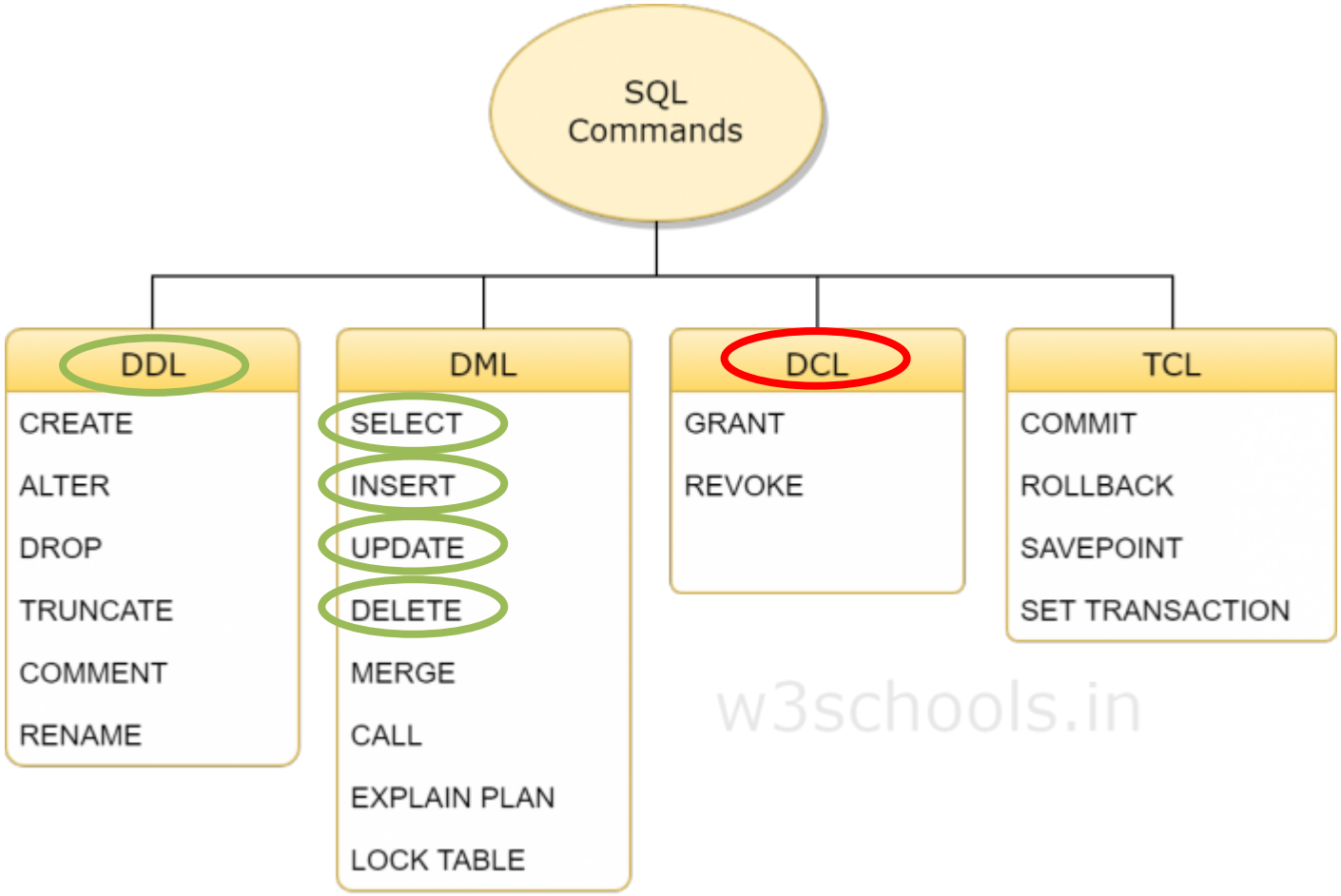
  TRUNCATE TABLE *instructor;*
- Difference to

  DELETE FROM *instructor;*
- ?

- **DELETE** keeps the table and deletes only the data
- **TRUNCATE** drops and re-creates the table
  - much faster
  - but cannot be undone
- **DELETE** is DML, **TRUNCATE** is DDL
  - Different rights may be necessary (see later!)

> **Description**
>
> `TRUNCATE TABLE` empties a table completely. It requires the `DROP` privilege (before 5.1.16, it required the `DELETE` privilege.) See `GRANT`.

Parts of SQL: The Big Picture



Source: https://www.w3schools.in/mysql/ddl-dml-dcl/

Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Recap: logical database model
  - The relations in the database and their attributes
- Views:
  - Virtual relations
  - Different from those in the database
  - But with the same data
  - ...hide data from users
- Example: instructors' names and departments without salaries, i.e.,

SELECT *ID*, *name*, *dept_name*
FROM *instructor*

Views

- Example: some users may see employees with salaries, others only without salary

- How about two tables
  - One with salaries
  - One without salaries
- ?

Defining Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- A view is defined using the **CREATE VIEW** statement:

  CREATE VIEW *v* **AS** < query expression >

  - <query expression> is any legal SQL expression
  - the view name is represented by *v*
- Once the view has been created
  - it can be addressed as *v* as any other relations
  - it will always contain the data read by the SQL expression
    - live, not at the time of definition!

Example Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Instructors without their salary

```
CREATE VIEW faculty AS
    SELECT ID, name, dept_name
    FROM instructor
```

- Using the view: find all instructors in the Biology department

```
SELECT name
FROM faculty
WHERE dept_name = 'Biology';
```

- Create a view of department salary totals

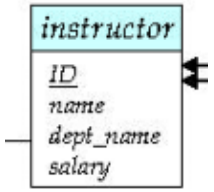
```
CREATE VIEW departments_total_salary(dept_name, total_salary) AS
    SELECT dept_name, SUM(salary)
    FROM instructor
    GROUP BY dept_name;
```

Updating Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Definition of a simple view
  (recap: instructors without salaries):

  CREATE VIEW *faculty* **AS**
  SELECT *ID*, *name*, *dept_name*
  FROM *instructor*



- Add a new tuple to *faculty* view which we defined earlier

  INSERT INTO *faculty* **VALUES** ('30765', 'Green', 'Music');

  What happens?

- This insertion must be represented by the insertion of the tuple

  ('30765', 'Green', 'Music', NULL)

  into the *instructor* relation

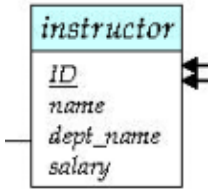
This can only work if salary is not defined as **NOT NULL**!

Updating Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Consider the view

  CREATE VIEW *biology_faculty* **AS**
  SELECT *ID,name*
  FROM *faculty*
  WHERE *dept_name* = ‘Biology’;



- and

  INSERT INTO biology_faculty
  VALUES (43278,‘Smith’);

- What would you expect?
  - Would this lead to

    INSERT INTO *instructor* **VALUES** (43278,’Smith’,’Biology’,NULL);
    ?

Updating Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Most **WHERE** constraints cannot be translated into a value to insert
  - Which?
- Consider

  WHERE *dept_name* = ‘Biology’ **or** *dept_name* = ‘Physics’
- or

  WHERE *salary* > 50000

- Hence, **WHERE** clauses are typically not translated into a value

Updating Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Other example used before

  CREATE VIEW *departments_total_salary*(*dept_name*, *total_salary*) **AS**
  SELECT *dept_name*, **SUM** (*salary*)
  FROM *instructor*
  GROUP BY *dept_name*;

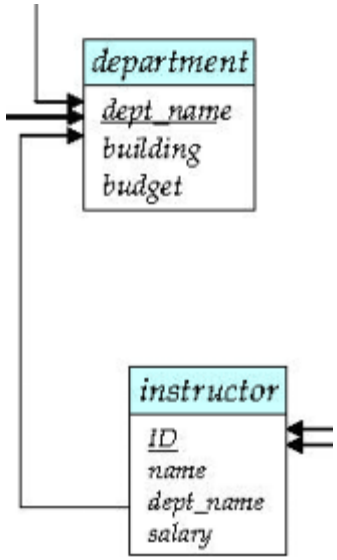
- What should happen upon

  UPDATE *departments_total_salary*
  SET *total_salary* = *total_salary* * 1.05
  WHERE *dept_name* = “Comp. Sci.”;
  ?

Updating Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **CREATE VIEW** *instructor_info* **AS**
  SELECT *ID*, *name*, *building*
  FROM *instructor*, *department*
  WHERE *instructor.dept_name= department.dept_name*;
- **INSERT INTO** *instructor_info* **VALUES** (’69987’, ’White’, ’Taylor’);
  - which department, if multiple departments are in Taylor?
  - what if no department is in Taylor?



Updateable Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- A view is called *updateable* if
  - The **FROM** clause has only one database relation
  - The **SELECT** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **DISTINCT** specification
  - Any attribute not listed in the **SELECT** clause can be set to null
  - The query does not have a **GROUP BY** or **HAVING** clause

- Most DMBS only allow updates on such views!

Materialized vs. Non-Materialized Views

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Normal views are not materialized
  - When issuing a **SELECT** against a view, the underlying data is created on the fly
  - Pro: guarantees recent and non-redundant data, saves space
  - Con: some views may be expensive to compute (e.g., extensive use of aggregates)
- **Materializing a view**: create a physical table containing all the tuples in the result of the query defining the view
  - If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated

Integrity Constraints

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Data errors may occur due to, e.g.,
  - Accidental wrong entries in form fields
  - Faulty application program code
  - Deliberate attacks
- Integrity constraints
  - guard against damage to the database
  - ensuring that authorized changes to the database do not result in a loss of data consistency
- Examples
  - A checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $20.00 an hour
  - A customer must have a (non-null) phone number

Integrity Constraints on a Single Relation

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- We have already encountered
  - **NOT NULL**
  - **PRIMARY** and **FOREIGN KEY**
- We will get to know
  - **UNIQUE**
  - **CHECK** (P), where P is a predicate

NOT NULL and UNIQUE Constraints

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **NOT NULL**
  - Declare *name* and *budget* to be **NOT NULL**

    name **VARCHAR**(20) **NOT NULL**
    budget **NUMERIC**(12,2) **NOT NULL**
- **UNIQUE** (A_1, A_2, …, A_m)
  - The unique specification states that the attributes A_1, A_2, … A_m form a candidate key
  - Candidate keys are permitted to be null (in contrast to primary keys)

The CHECK Constraint

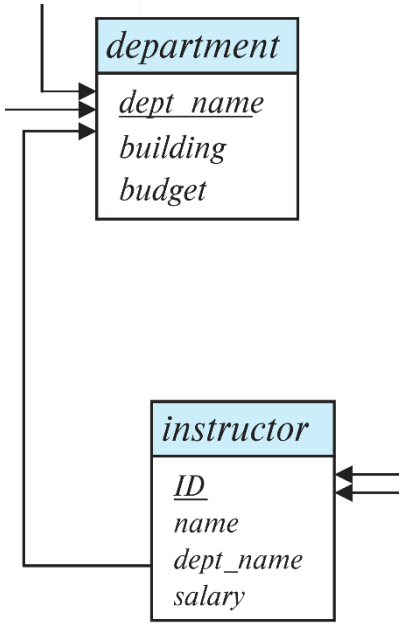
UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **CHECK** (P)
  - where P is a predicate
- Example: ensure that semester is either fall or spring

```
CREATE TABLE section (
        course_id VARCHAR (8),
        sec_id VARCHAR (8),
        semester VARCHAR (6),
        ....
        CHECK (semester IN ('Fall', 'Spring'))
      );
```

Foreign Keys and Referential Integrity

UNIVERSITY OF MANNHEIM
Data and Web Science Group



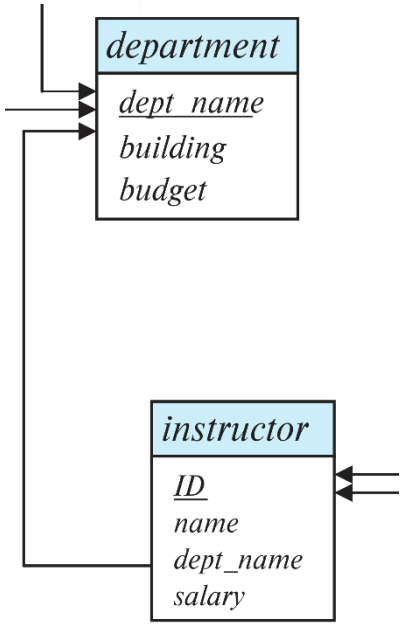
- Example:
  - instructors have a department
  - **each** department should also appear in the *department* relation
    - Otherwise error message is shown

CREATE TABLE *instructor* (
ID **CHAR**(5),
name **VARCHAR**(20),
dept_name **VARCHAR**(20),
salary **NUMERIC**(8,2))
PRIMARY KEY (*ID*),
FOREIGN KEY (*dept_name*)
REFERENCES *department(dept_name)*
);

Foreign key can also be NULL, then no check is peformed

Cascading Actions in Referential Integrity

- Example:
  - instructors have a department
  - **each** department should also appear in the *department* relation
- How to *ensure* referential integrity?
  - i.e., what happens if a department is deleted from the *department* relation
- Possible approaches
  - Reject the deletion — default action
  - Delete all instructors as well
  - Set the department of those instructors to **null**



Cascading Actions in Referential Integrity

- Cascading updates
  - If a foreign key is changed (e.g., renaming a department)
  - ...then rename in all referring relations
- Cascading deletions
  - If a foreign key is deleted (e.g., deleting a department)
  - ...then delete all rows in referring relations
- **CREATE TABLE** *instructor* (

  …

  dept_name **VARCHAR**(20),

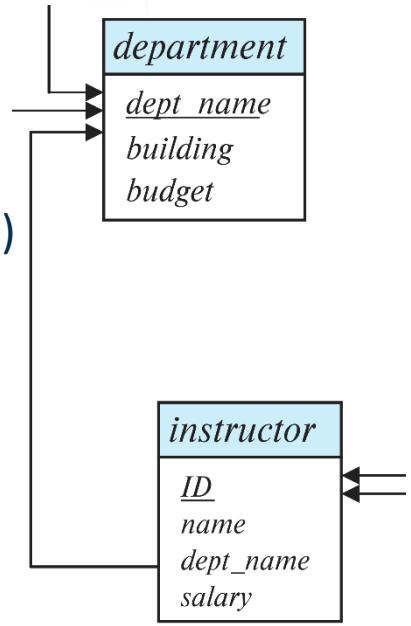
  FOREIGN KEY (*dept_name*) **REFERENCES** *department*

  ON DELETE CASCADE

  ON UPDATE CASCADE,

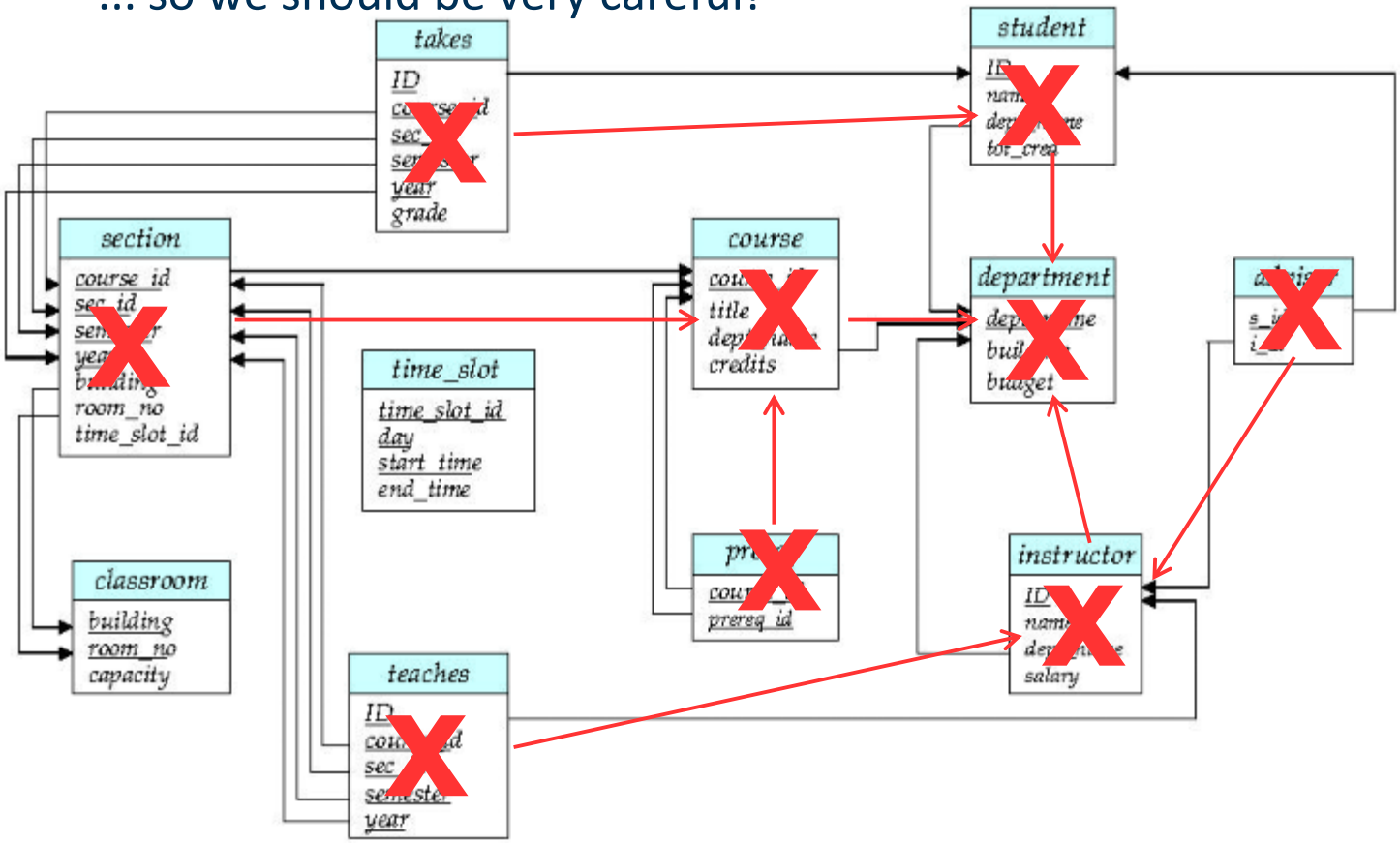
  . . .

)



Cascading Actions in Referential Integrity

- Cascading deletions may run over several tables
  - ... so we should be very careful!



Cascading Actions in Referential Integrity

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **SET NULL** for updates
  - If a foreign key is changed (e.g., renaming a department)
  - ...then set null for all referring relations
- **SET NULL** for deletions
  - If a foreign key is deleted (e.g., deleting a department)
  - ...then set null in referring relations
- **CREATE TABLE** *instructor* (

```
    …
    dept_name VARCHAR(20),
    FOREIGN KEY (dept_name) REFERENCES department
            ON DELETE SET NULL
            ON UPDATE SET NULL,
    . . .
)
```

Different behavior for update and delete is also possible

Authorization

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Rights for accessing a database may differ
  - Only administrators may change the schema
- Rights for accessing a database can be very fine grained
  - Not everybody may see a persons' salary
  - Not everybody may alter a persons' salary
  - Nobody may alter their own salary
  - Special restrictions may apply for entering salaries over a certain upper bound
  - …

Authorization Specification in SQL

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- The **GRANT** statement is used to confer authorization

  GRANT <privilege list>

  ON <relation name or view name>

  TO <user list>

  - a user-id
  - **PUBLIC**, which allows all valid users the privilege granted
  - A role (more on this later)

- Granting a privilege on a view does not imply granting any privileges on the underlying relations
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator)

Privilege Definition in SQL

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **SELECT**: allows read access to relation, or the ability to query using the view
  - Example: grant users Stephen and Mary **SELECT** authorization on the *instructor* relation:

    GRANT SELECT ON *instructor* **TO** *Stephen, Mary*
- **INSERT**: the ability to insert tuples
- **UPDATE**: the ability to update using the SQL update statement
- **DELETE**: the ability to delete tuples.
- **ALL PRIVILEGES**: used as a short form for all the allowable privileges

Revoking Privileges

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- The **REVOKE** statement is used to revoke authorization.

  REVOKE <privilege list>

  ON <relation name or view name>

  FROM <user list>

  ALL to revoke all privileges the revokee may hold

- Example:

  REVOKE SELECT ON *branch* **FROM** *Stephen, Mary*

- If <user list> includes **PUBLIC,**
  all users lose the privilege except those granted it explicitly
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation
- All privileges that depend on the privilege being revoked are also revoked

Revoking Privileges

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Scenario 1:
  - **GRANT SELECT ON** instructor **TO** John, Mary
  - **REVOKE SELECT ON** instructor **FROM** John
  - → Mary retains right
- Scenario 2:
  - **GRANT SELECT ON** instructor **TO PUBLIC**
  - **GRANT ALL ON** instructor **TO** John
  - **REVOKE ALL ON** instructor from **PUBLIC**
  - → John retains right, since he has been granted the right explicitly

Roles

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Databases may have many users
  - e.g., all students and employees of a university
- Managing privileges for all those individually can be difficult
  - User groups (also called: roles) are more handy
  - Example roles
    - Student
    - Instructor
    - Secretary
    - Dean
    - ...

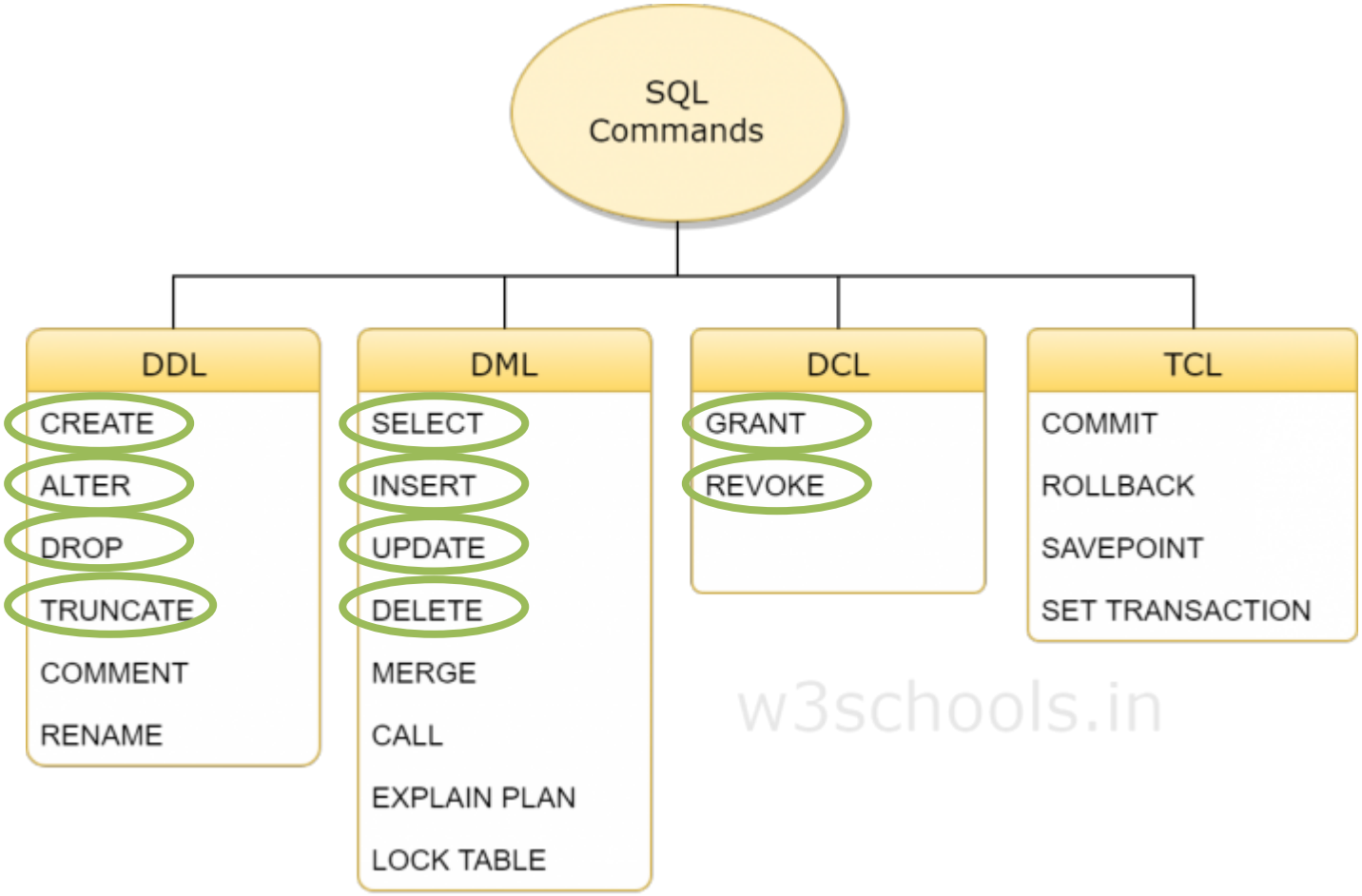
Roles

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Creating roles and assigning them to individual users
  - **CREATE ROLE** instructor;
  - **GRANT** *instructor* **TO** Amit
- Granting privileges to roles
  - **GRANT SELECT ON** *takes* **TO** *instructor*

- Roles can form hierarchies
  - i.e., a role inherits from other roles
  - **CREATE ROLE** *teaching_assistant*
  - **GRANT** *teaching_assistant* **TO** *instructor*
  - *Instructor* inherits all privileges of *teaching_assistant*

Parts of SQL: The Big Picture



Source: https://www.w3schools.in/mysql/ddl-dml-dcl/

Wrap-up

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Today, we have seen
  - How to manipulate data in databases
  - i.e., **INSERT**, **UPDATE**, and **DELETE** statements
- Views
  - are used to provide different subsets
    and/or aggregations of data
  - updateable views
  - materialized views

Wrap-up

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Integrity constraints
  - unique and not null constraints
  - cascading updates and deletions
- Access rights
  - can be fine grained
  - can be bound to user groups and roles
  - roles may inherit from each other

Questions?