

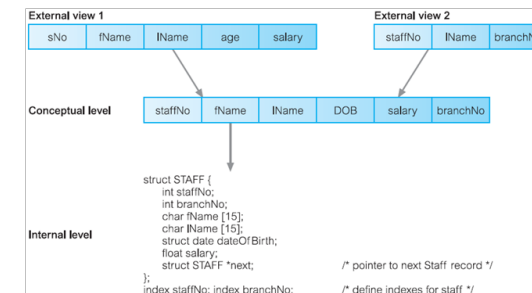
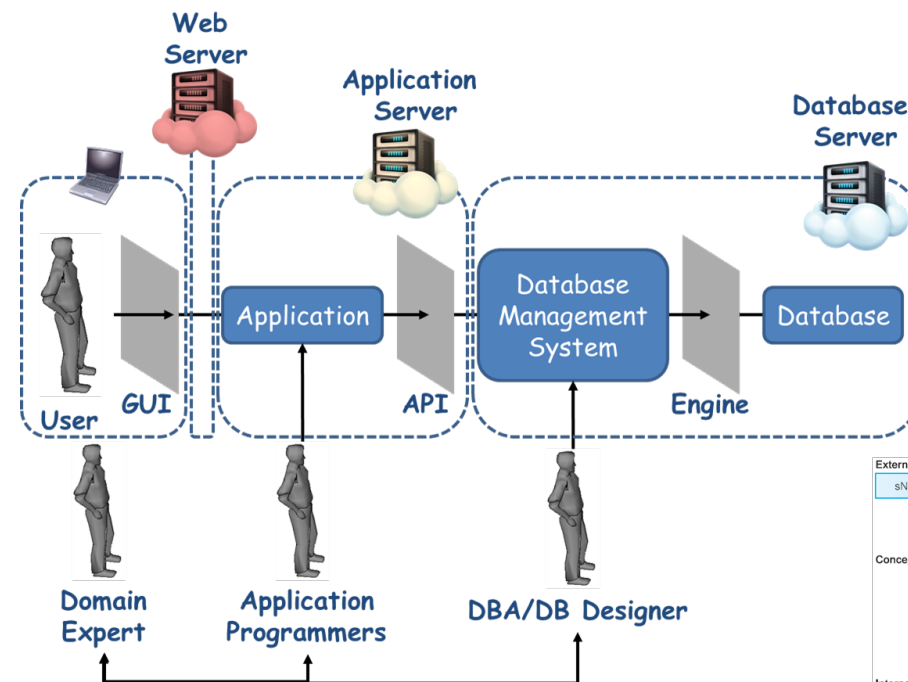
Web Systems Fundamentals and Databases (Grundläggande webbsystem och databaser) DI4020 - 11hp

Lectures 6, 7 and 8

Wagner Ourique de Morais

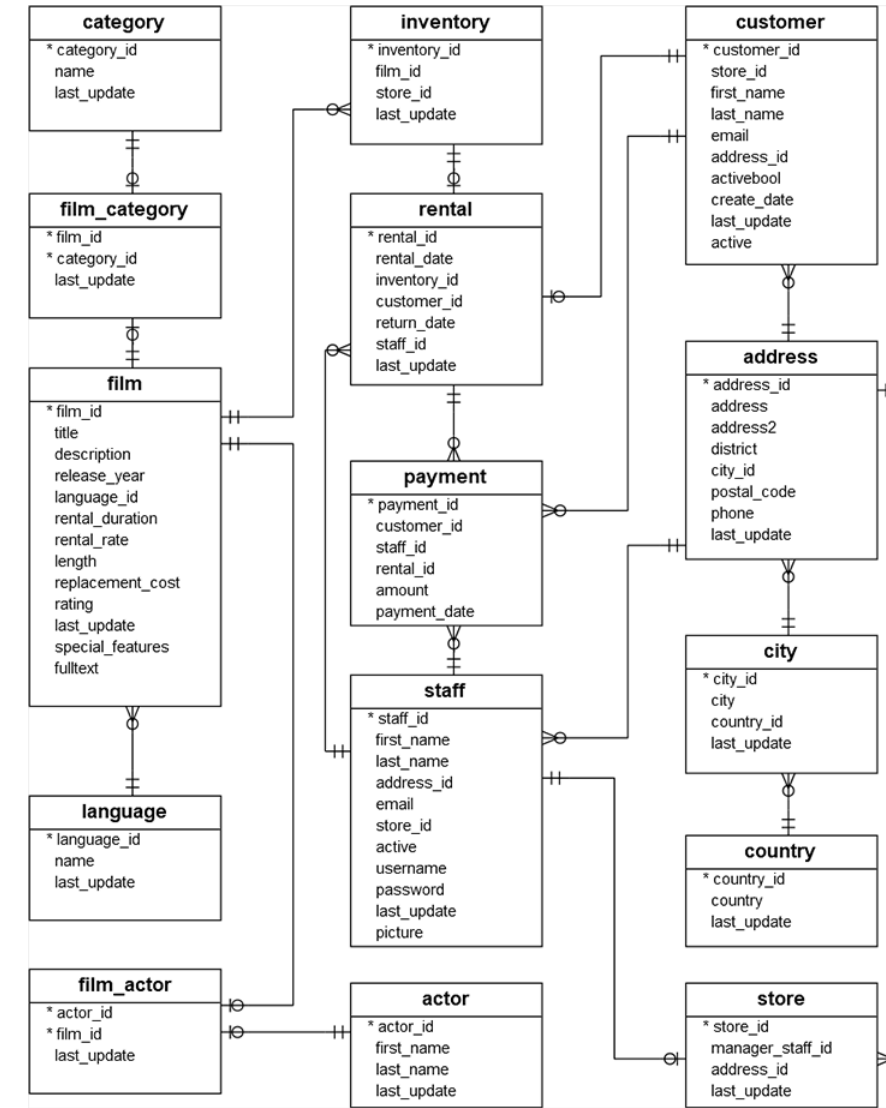
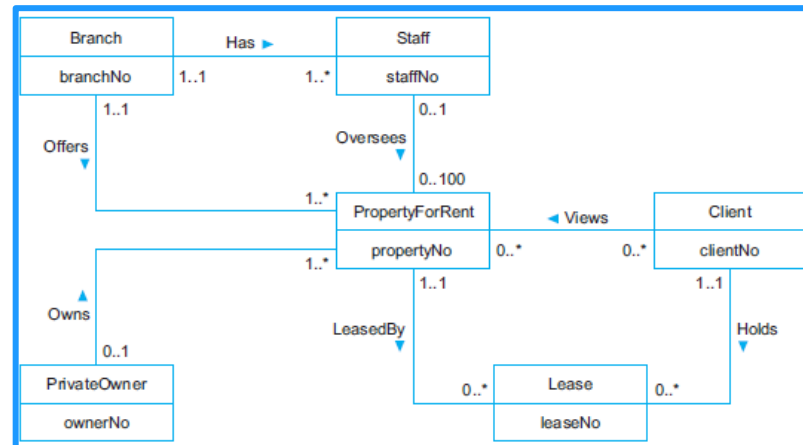
Previously in the course

- Course Introduction
- Lecture 1 and 2
 - Introduction to the field of database systems and database design.

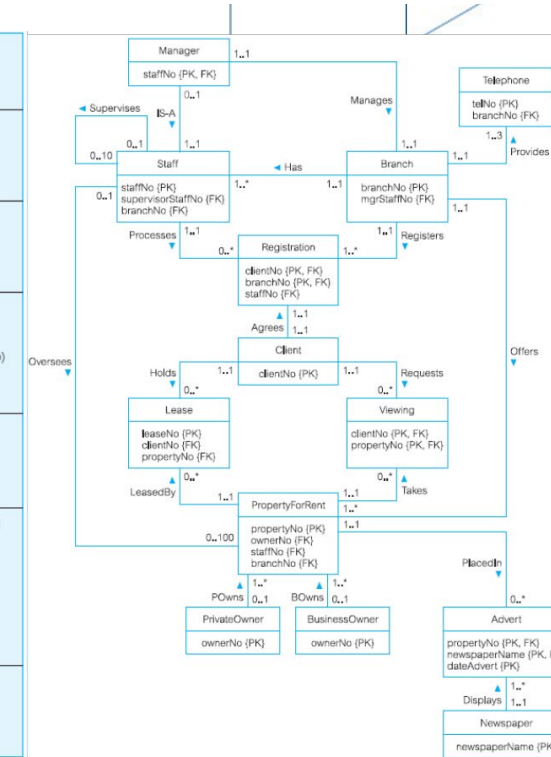


Previously in the course

- Lectures 3 and 4
 - Types of database design
 - Database design process
 - Conceptual, logical and physical design.
 - Conceptual database design
 - Build conceptual data model independent of all physical considerations.
 - Entity-relationship (ER) data model and ER diagrams

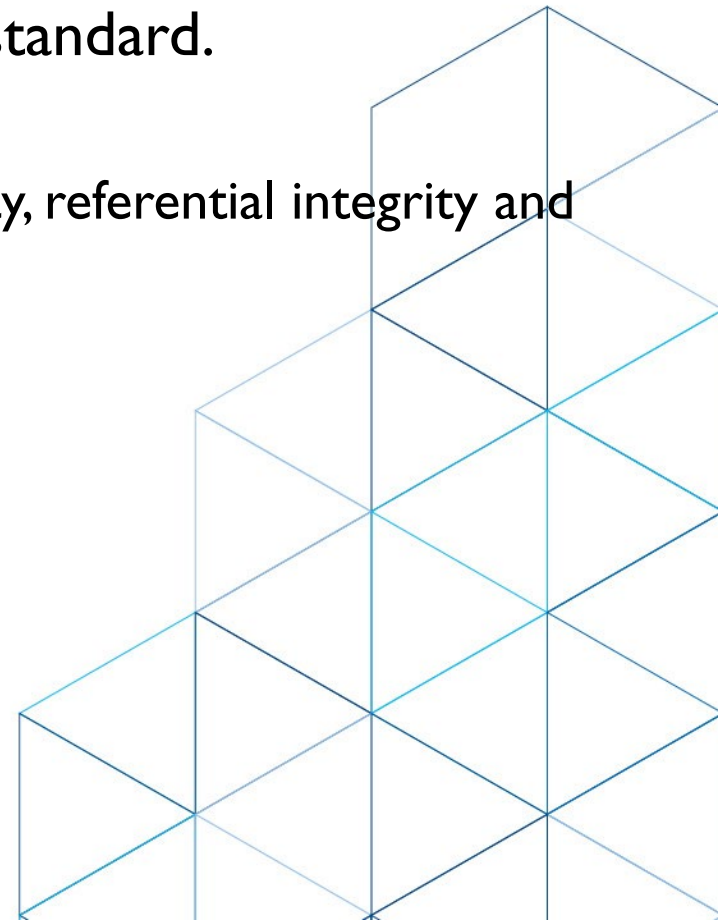


- Normalization



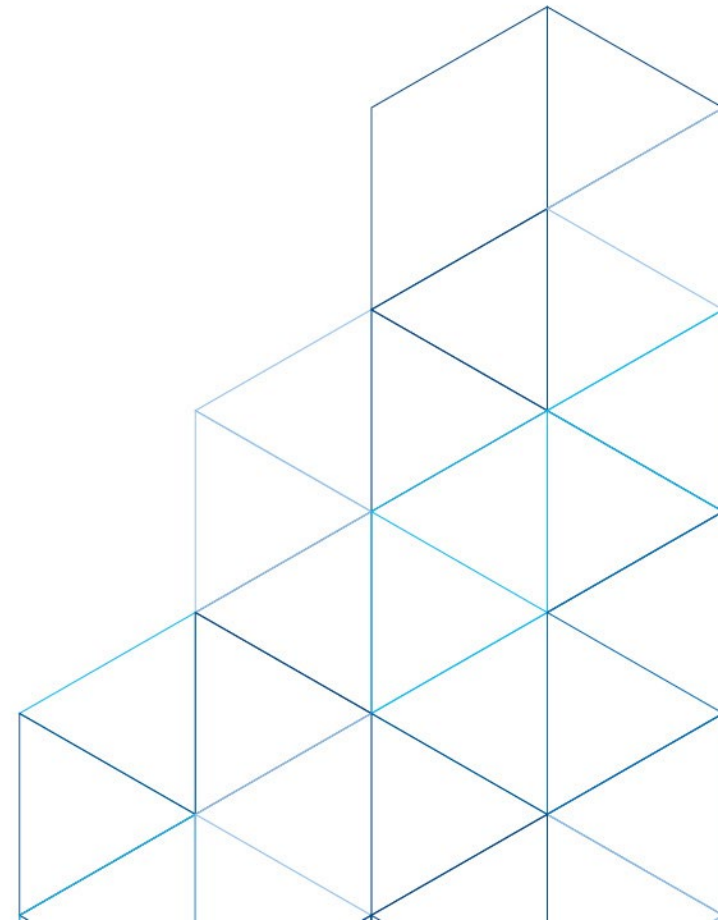
Objectives

- Chapter 6 introduces the data manipulation statements of the SQL standard: SELECT, INSERT, UPDATE, and DELETE.
- Chapter 7 covers the main data definition facilities of the SQL standard.
 - Including SQL data types and data definition statements,
 - Integrity constraints (required data, domain constraints, entity integrity, referential integrity and general constraints)
 - Creating views.



Chapter 6 - SQL – Data Manipulation

- Purpose and importance of SQL.
- How to retrieve data from database using SELECT and:
 - Use compound WHERE conditions.
 - Sort query results using ORDER BY.
 - Use aggregate functions.
 - Group data using GROUP BY and HAVING.
 - Use subqueries.
 - Join tables together.
 - Perform set operations (UNION, INTERSECT, EXCEPT).
- How to update database using INSERT, UPDATE, and DELETE.



Chapter 7 - SQL – Data Definition

- Data types supported by SQL standard.
- Data definition
- Purpose of integrity enhancement feature of SQL.
- How to define integrity constraints using SQL.
- How to use the integrity enhancement feature in the **CREATE** and **ALTER TABLE** statements.



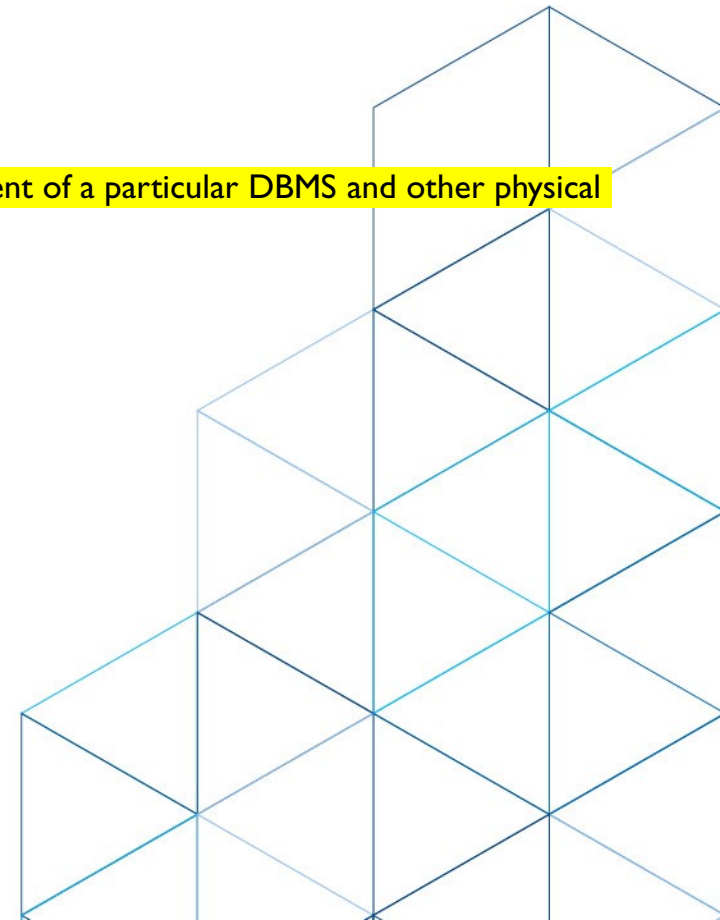
...more specifically

- Physical database design
- Introduce relational database programming using Structured Query Language (SQL)
- Understand the purpose and importance of SQL
- Understand the SQL syntax and how to write SQL commands
- Understand how to use SQL to perform data definition
 - CREATE and DROP a database
 - CREATE, ALTER and DROP database objects (TABLES, VIEWS, ...)
 - Integrity constraints
- Understand how to use SQL to perform data manipulation
 - SELECT, INSERT, UPDATE and DELETE



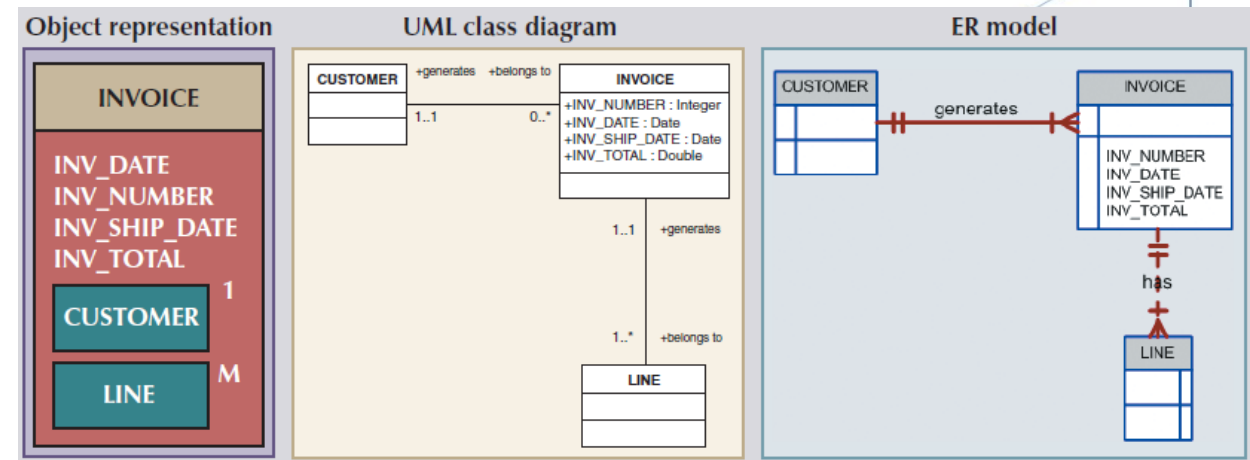
Overview of the Database Design process

- Conceptual database design
 1. The process of constructing a model of the data used in an enterprise, independent of all physical considerations.
 - Identify entity types, relationship types and attributes.
 - Determine attribute domains
 - Determine candidate, primary, and alternate key attributes
 - Check model for redundancy
 - Validate conceptual data model against user transactions
 - Review conceptual data model with user
- Logical database design for the relational model
 2. The process of constructing a model of the data based on a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.
 - Derive relations for logical data model
 - Validate relations using normalization
 - Validate relations against user transactions
 - Check integrity constraints
 - Review logical data model with user
- Physical database design for relational databases
 3. Translate logical data model for target DBMS.
 - Design base relations, representation of derived data and general constraints
 4. Design file organizations and indexes
 5. Design user views
 6. Design security mechanisms
 7. Consider the introduction of controlled redundancy
 8. Monitor and tune the operational system



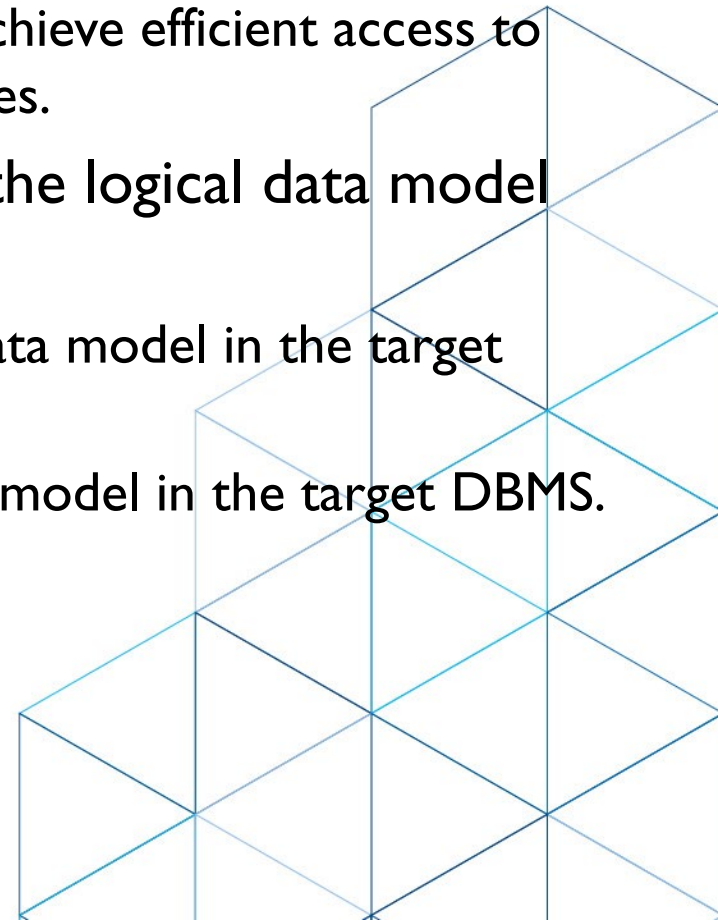
Data modeling

- A data model is a relatively simple representation, usually graphical, of more complex real-world data structures.
- Data model vs database model
 - Database model refers to the implementation of a data model in a specific database system.
- Data modeling is an iterative, progressive process.
- Data models can facilitate interaction among the designer, the applications programmer, and the end user.



Physical database design

- The process of producing a description of the implementation of the database on secondary storage
 - Describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.
- We will focus on producing a relational database schema from the logical data model that can be implemented in the target DBMS.
 - Decide how to represent the base relations identified in the logical data model in the target DBMS.
 - Decide how to represent any derived data present in the logical data model in the target DBMS.
 - Design the general constraints for the target DBMS.



Database Management Systems

- A DBMS is a specialized software used to manage databases.
 - Enables users to define, create, maintain, and control access to the database.
- All DBMSs are capable of
 - managing large amounts of data,
 - maintaining data integrity,
 - responding to many queries,
 - creating indexes and triggers, and more
- Examples
 - PostgreSQL, Oracle Database, IBM DB2, Microsoft SQL Server, MySQL, SQLite



Relational databases

- A database in a Relational DBMS is composed of one or more tables.
- A table (entity) is a two-dimensional container for data that consists of records (rows);
- Each record has the same number of attributes (columns or fields), which contain the actual data.
- Each table will have one special field called a primary key that is used to uniquely identify each record in a table.
- A relationship is an association between tables.

ArtWorkID	Title	Artist	YearOfWork
345	The Death of Marat	David	1793
400	The School of Athens	Raphael	1510
408	Bacchus and Ariadne	Titian	1520
425	Girl with a Pearl Earring	Vermeer	1665
438	Starry Night	Van Gogh	1889

ArtWorkID	Title	ArtistID	YearOfWork
345	The Death of Marat	15	1793
400	The School of Athens	37	1510
408	Bacchus and Ariadne	25	1520
425	Girl with a Pearl Earring	22	1665
438	Starry Night	43	1889

Foreign key

ArtistID	Artist
15	David
22	Vermeer
25	Titian
37	Raphael
43	Van Gogh

Primary key

Relationships between Tables

- Tables that are linked via foreign keys are said to be in a relationship:
 - one-to-many relationship
single record in Table A can have one or more matching records in Table B
 - many-to-many relationship
many-to-many relationships are usually implemented by using an intermediate table with two one-to-many relationships
 - One-to-one relationship
typically used for security or performance reasons. (Could be 1 table)

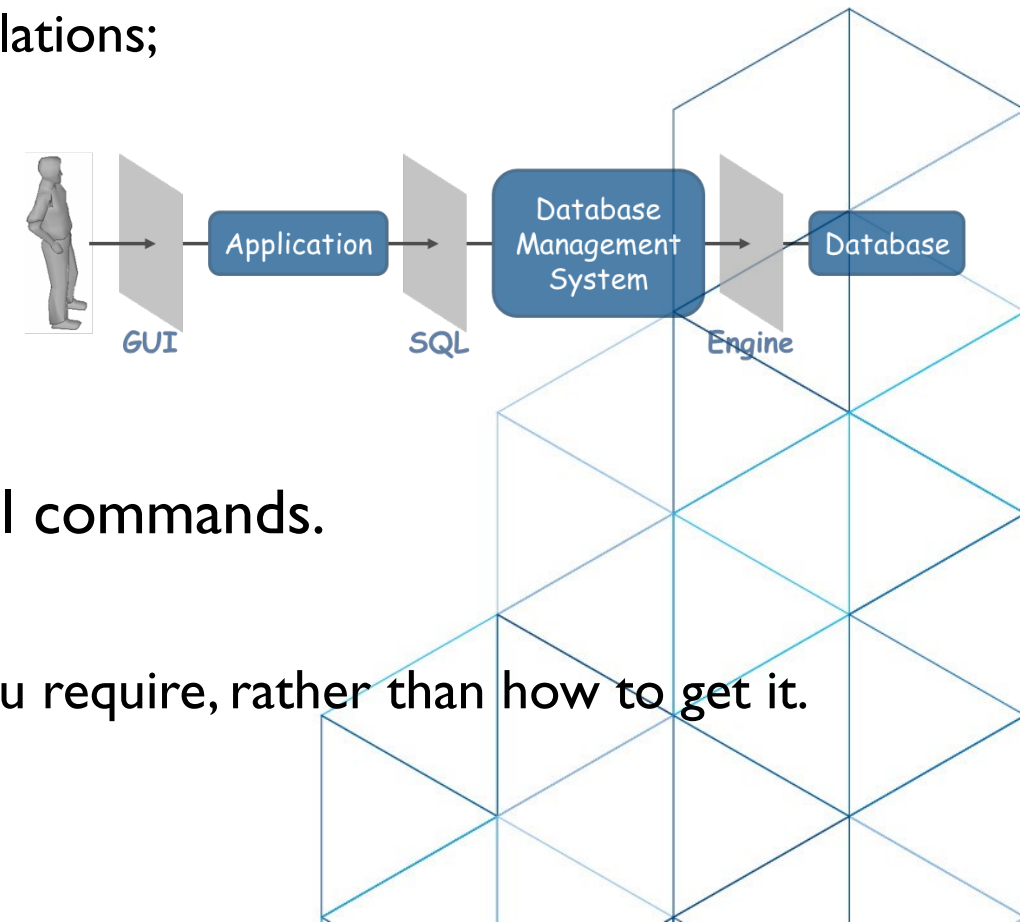


Comments or Questions?



SQL - Pronounced sequel (or sss queue elle)

- Ideally, database language should allow user to:
 - create the database and relation structures;
 - perform insertion, modification, deletion of data from relations;
 - perform simple and complex queries.
- SQL is the standard relational database language.
 - 2 major components:
 - DDL for defining database structure.
 - A DML for retrieving, inserting, updating and deleting data.
- Until SQL:1999, SQL did not contain flow of control commands.
- SQL is relatively easy to learn:
 - It is non-procedural, i.e. you specify what information you require, rather than how to get it.



DATA DEFINITION

- CREATE tables, views, database ...
- ADD or ALTER columns
- DROP tables, views, ...

C
O
L
S

ID	Fname	Lname	email	
1	Wagnre	Morais		
2	Jesper	Hakerod		

rows

DATA MANIPULATION

- SELECT
- INSERT
- UPDATE
- DELETE

History of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' (SEQUEL).
 - A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
 - Still pronounced 'see-quel', though official pronunciation is 'S-Q-L'.
 - IBM subsequently produced a prototype DBMS called System R, based on SEQUEL/2.
 - Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.
-
- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.
 - In 1987, ANSI and ISO published an initial standard for SQL.
 - In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.
 - In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
 - In 1999, SQL:1999 was released with support for object-oriented data management.
 - In late 2003, SQL:2003 was released.
 - In summer 2008, SQL:2008 was released.
 - In late 2011, SQL:2011 was released.



Objectives of SQL (cont.)

- Consists of standard English words:

```
CREATE TABLE Staff(  
    staffNo VARCHAR(5),  
    lName SMALL_INT,  
    salary DECIMAL(7,2));
```

```
INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
```

```
SELECT staffNo, lName, salary  
FROM Staff  
WHERE salary > 10000;
```



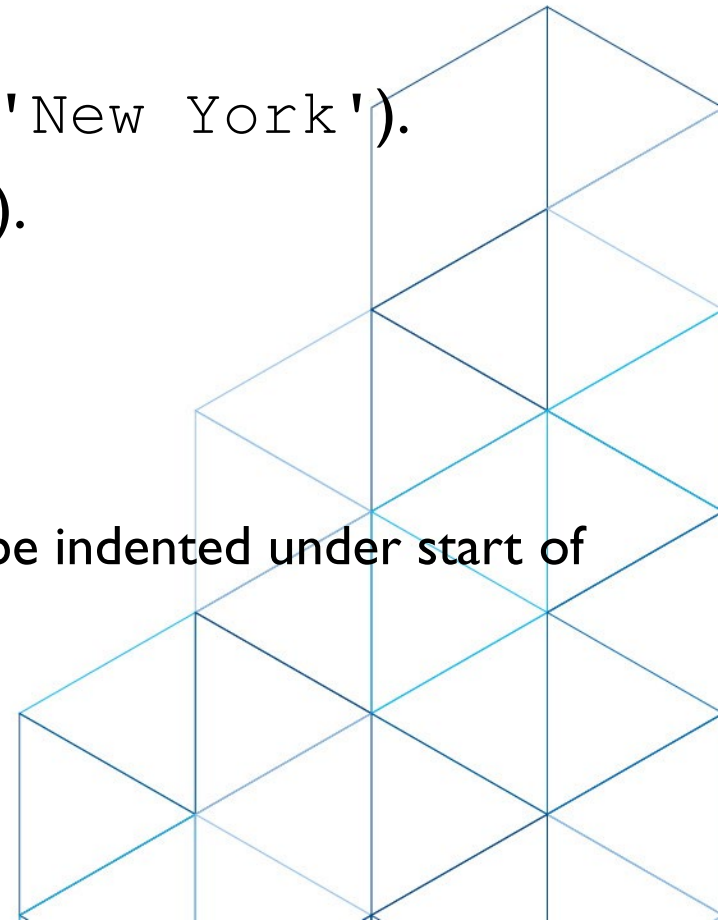
Writing SQL Commands or statements

- Use extended form of BNF notation:
 - Upper-case letters represent reserved words.
 - Lower-case letters represent user-defined words.
 - | indicates a choice among alternatives.
 - Curly braces indicate a required element.
 - Square brackets indicate an optional element.
 - ... indicates optional repetition (0 or more).
- SQL statement consists of reserved words and user-defined words.
 - Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
 - User-defined words are made up by user and represent names of various database objects such as relations, columns, views.

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]  
      [ * | expression [ [ AS ] output_name ] [, ...] ]  
      [ FROM from_item [, ...] ]  
      [ WHERE condition ]  
      [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]  
      [ HAVING condition ]  
      [ WINDOW window_name AS ( window_definition ) [, ...] ]  
      [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select  
      [ ORDER BY expression [ ASC | DESC | USING operator ] [, ...]  
      [ LIMIT { count | ALL } ]  
      [ OFFSET start [ ROW | ROWS ] ]
```

Writing SQL Commands or statements

- Most components of an SQL statement are case insensitive, except for literal character data.
 - 'hello world' is different than 'Hello World'
- All non-numeric literals must be enclosed in single quotes (e.g. 'New York').
- All numeric literals must not be enclosed in quotes (e.g. 650.00).
- More readable with indentation and lineation:
 - Each clause should begin on a new line.
 - Start of a clause should line up with start of other clauses.
 - If clause has several parts, should each appear on a separate line and be indented under start of clause.



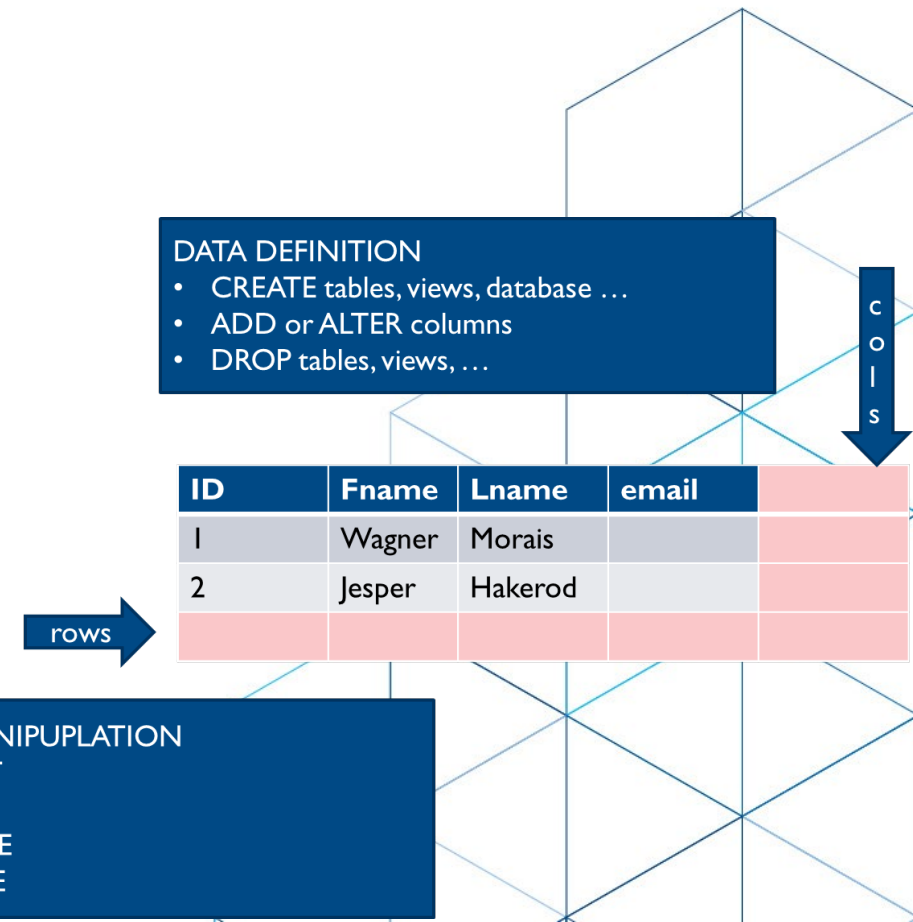
Data Definition Language (DDL)



Data Definition Language (DDL)

- DDL statements used for creating, altering and dropping databases, tables, relationships, views, and other structures.

```
CREATE TABLE customer (  
  customer_id integer DEFAULT nextval('customer_customer_id_seq'::regclass) NOT NULL,  
  store_id smallint NOT NULL,  
  first_name character varying(45) NOT NULL,  
  last_name character varying(45) NOT NULL,  
  email character varying(50),  
  address_id smallint NOT NULL,  
  activebool boolean DEFAULT true NOT NULL,  
  create_date date DEFAULT ('now'::text)::date NOT NULL,  
  last_update timestamp without time zone DEFAULT now(),  
  active integer  
);  
ALTER TABLE customer  
  ADD CONSTRAINT customer_pkey PRIMARY KEY (customer_id);  
ALTER TABLE customer  
  ADD CONSTRAINT customer_address_id_fkey FOREIGN KEY (address_id) REFERENCES  
address(address_id) ON UPDATE CASCADE ON DELETE RESTRICT;
```



Data Definition Statements

- SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.

- Main SQL DDL statements are:

CREATE DATABASE

CREATE SCHEMA

CREATE/ALTER DOMAIN

CREATE/ALTER TABLE

CREATE VIEW

DROP DATABASE

DROP SCHEMA

DROP DOMAIN

DROP TABLE

DROP VIEW

ALTER DATABASE

ALTER SCHEMA

ALTER DOMAIN

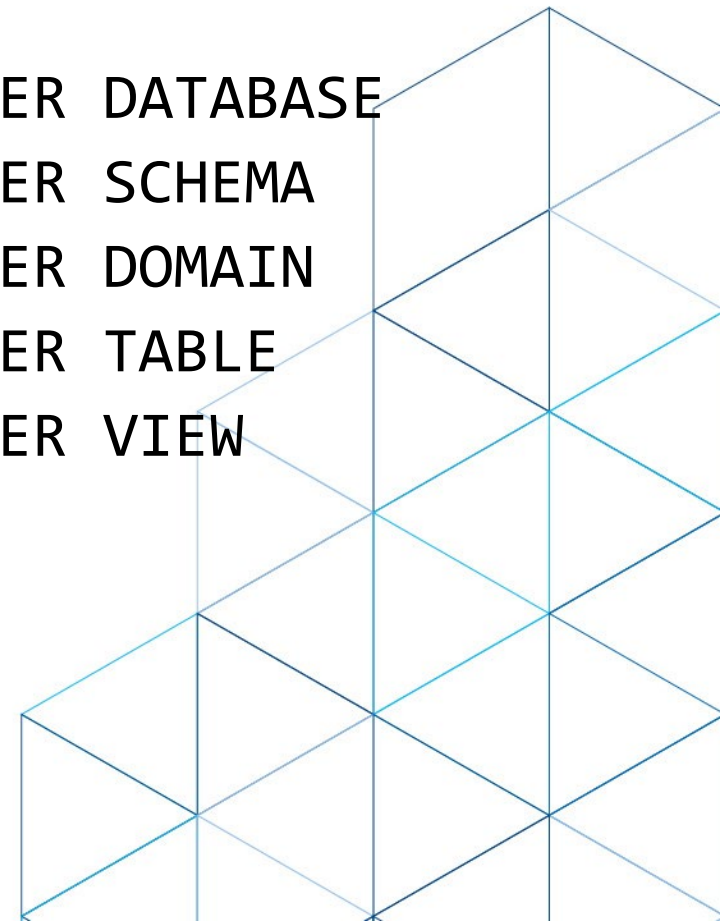
ALTER TABLE

ALTER VIEW

- Many DBMSs also provide:

CREATE INDEX

DROP INDEX



Data Definition

- Relations and other database objects exist in an environment.
- Each environment contains one or more catalogs, and each catalog consists of set of schemas.
- Schema is named collection of related database objects.
- Objects in a schema can be tables, views, domains, assertions, collations, translations, and character sets. All have same owner.



ISO SQL Data Types

DATA TYPE	DECLARATIONS				
boolean	BOOLEAN				
character	CHAR	VARCHAR			
bit [†]	BIT	BIT VARYING			
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT	BIGINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION		
datetime	DATE	TIME	TIMESTAMP		
interval	INTERVAL				
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT		

[†]BIT and BIT VARYING have been removed from the SQL:2003 standard.

Integrity constraints

- **Required data**

```
firstName VARCHAR(100) NOT NULL
```

- **Domain constraints**

```
sex CHAR NOT NULL CHECK (sex IN ('M', 'F'))
```

- **Entity integrity**

- Primary key of a table must contain a unique, non-null value for each row.

```
PRIMARY KEY(staffNo)
```

```
PRIMARY KEY(clientNo, propertyNo)
```

- **General constraints**

- CHECK/UNIQUE in CREATE and ALTER TABLE.



Integrity constraints

- Referential integrity

- if FK contains a value, that value must refer to existing row in parent table.

- FOREIGN KEY (address_id) REFERENCES address(address_id)

- Any INSERT/UPDATE attempting to create FK value in child table without matching CK value in parent is rejected.

- Action taken attempting to update/delete a CK value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:

- CASCADE: Delete row from parent and delete matching rows in child, and so on in cascading manner.
 - SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are NOT NULL.
 - SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.
 - NO ACTION: Reject delete from parent. Default.

- FOREIGN KEY (address_id) REFERENCES address(address_id) ON UPDATE CASCADE ON DELETE RESTRICT;

CREATE TABLE

- Creates a table with one or more columns of the specified dataType.
- With NOT NULL, system rejects any attempt to insert a null in the column.
- Can specify a DEFAULT value for the column.
- Primary keys should always be specified as NOT NULL.
- FOREIGN KEY clause specifies FK along with the referential action.

```
CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] [...,]}
{[FOREIGN KEY (listOfFKColumns)
REFERENCES ParentTableName [(listOfCKColumns)],
[ON UPDATE referentialAction]
[ON DELETE referentialAction ]] [,...]}
{[CHECK (searchCondition)] [,... ]})
```

```
CREATE TABLE PropertyForRent (
propertyNo VARCHAR(5) NOT NULL,
rooms SMALLINT NOT NULL DEFAULT 4,
rent DECIMAL(6,2) NOT NULL, DEFAULT 600,
ownerNo VARCHAR(5) NOT NULL,
branchNo VARCHAR(5) NOT NULL,
PRIMARY KEY (propertyNo),
FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL ON
UPDATE CASCADE);
```

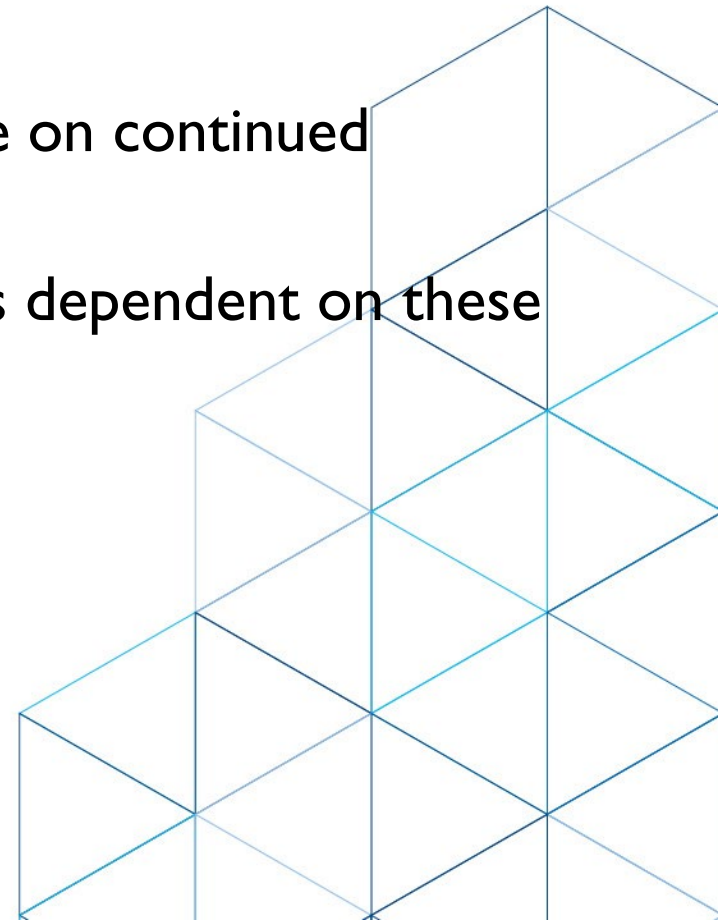

ALTER TABLE

- Add a new column to a table.
`ALTER TABLE table_name
ADD column_name datatype;`
- Drop a column from a table.
`ALTER TABLE table_name
DROP COLUMN column_name;`
- Add a new table constraint.
- Drop a table constraint.
`ALTER TABLE table_name
DROP CONSTRAINT constraint_name;`
- Set a default for a column.
`ALTER TABLE table_name
ALTER column_name SET DEFAULT 'F';`
- Drop a default for a column.



DROP TABLE

- `DROP TABLE TableName [RESTRICT | CASCADE]`
`DROP TABLE PropertyForRent;`
- Removes named table and all rows within it.
- With `RESTRICT`, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With `CASCADE`, SQL drops all dependent objects (and objects dependent on these objects).



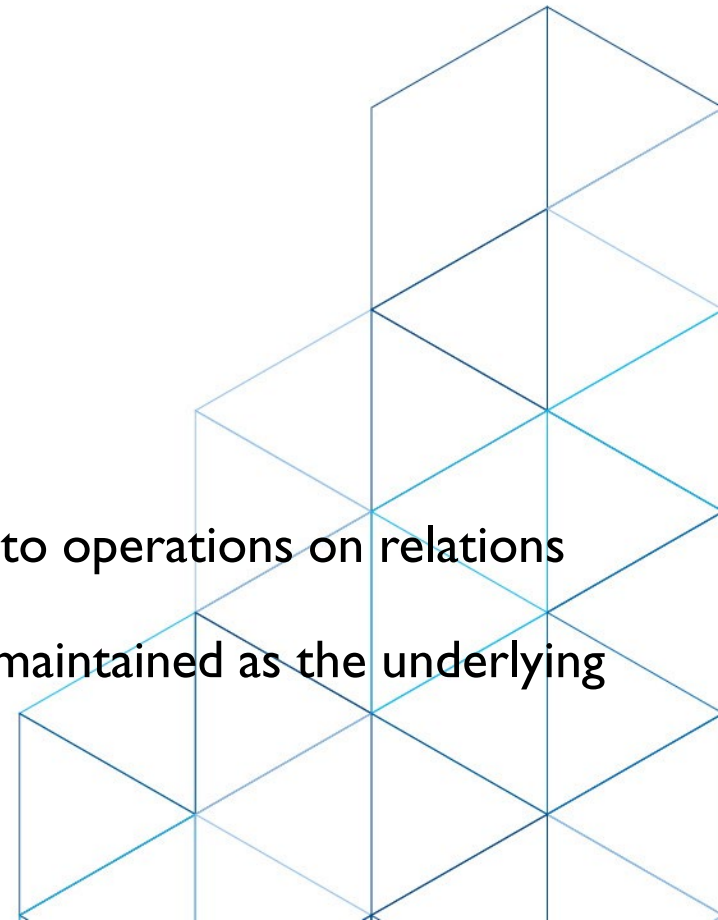
Views

- Dynamic result of one or more relational operations operating on base relations to produce another relation.
- Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.

```
CREATE VIEW ViewName [(newColName [,...])] AS  
    subselect; --defining query
```

```
CREATE VIEW view_name AS  
    SELECT column1, column2, ...  
    FROM table_name  
    WHERE condition;
```

- Contents of a view are defined as a query on one or more base relations.
- With view resolution, any operations on view are automatically translated into operations on relations from which it is derived.
- With view materialization, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.



Views

- Advantages of Views
 - Data independence
 - Currency
 - Improved security
 - Reduced complexity
 - Convenience
 - Customization
 - Data integrity

- Disadvantages of Views
 - Update restriction
 - Structure restriction
 - Performance



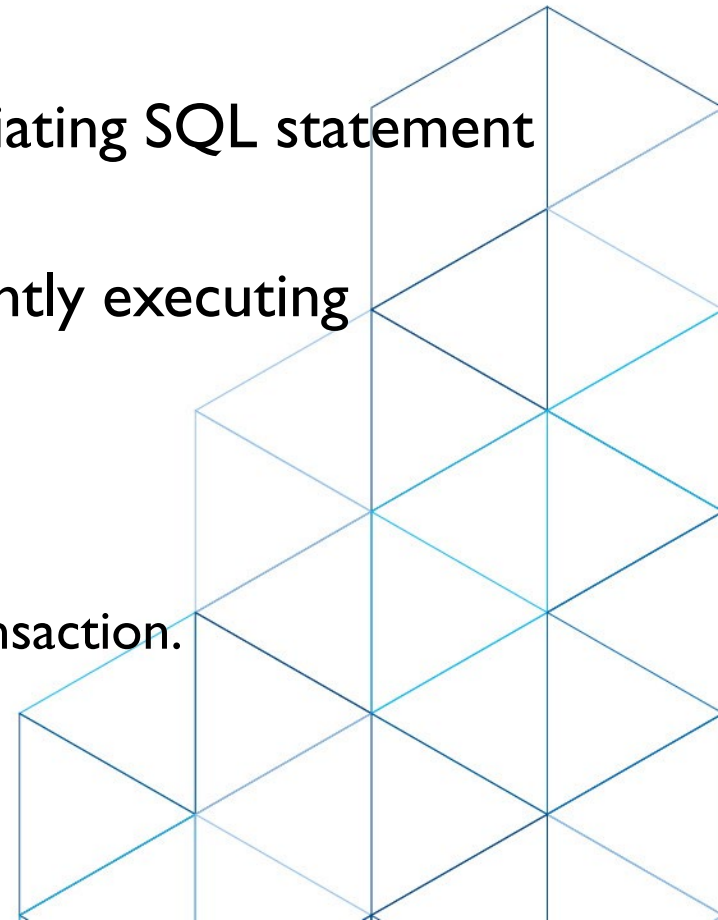
View Materialization

- View resolution mechanism may be slow, particularly if view is accessed frequently.
- View materialization stores view as temporary table when view is first queried.
- Thereafter, queries based on materialized view can be faster than recomputing view each time.
- Difficulty is maintaining the currency of view while base tables(s) are being updated.



Transactions

- SQL defines transaction model based on COMMIT and ROLLBACK.
- Transaction is logical unit of work with one or more SQL statements guaranteed to be atomic with respect to recovery.
- An SQL transaction automatically begins with a transaction-initiating SQL statement (e.g., SELECT, INSERT).
- Changes made by transaction are not visible to other concurrently executing transactions until transaction completes.
- Transaction can complete at least in two ways:
 - COMMIT ends transaction successfully, making changes permanent.
 - ROLLBACK aborts transaction, backing out any changes made by transaction.



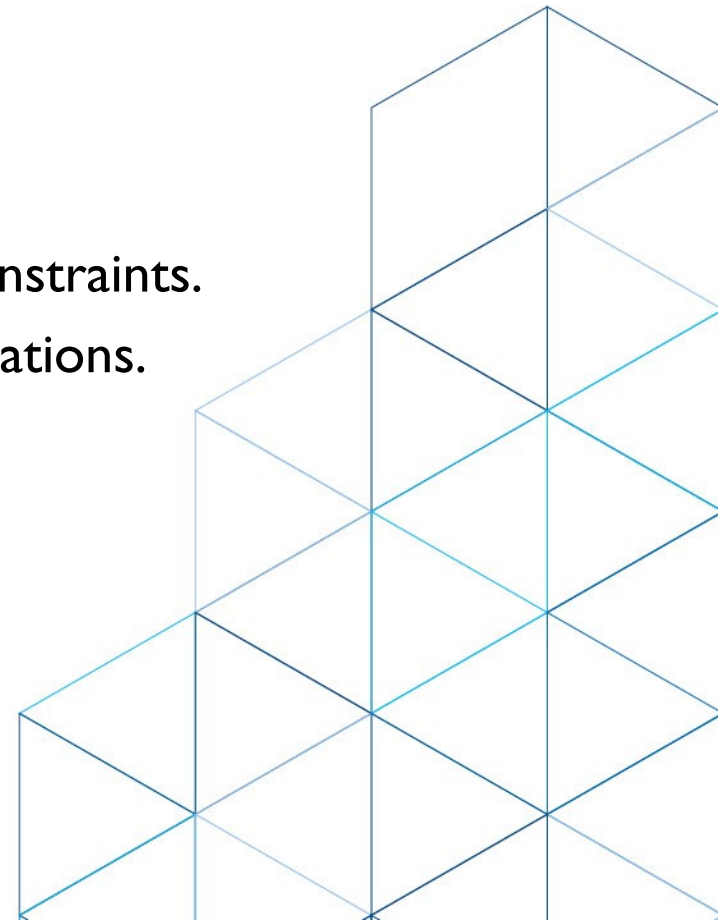
Access Control - Authorization Identifiers and Ownership

- Authorization identifier is normal SQL identifier used to establish identity of a user. Usually has an associated password.
- Used to determine which objects user may reference and what operations may be performed on those objects.
- Each object created in SQL has an owner, as defined in **AUTHORIZATION** clause of schema to which object belongs.
- Owner is only person who may know about it.



Priviledges

- Actions user permitted to carry out on given base table or view:
 - SELECT Retrieve data from a table.
 - INSERT Insert new rows into a table.
 - UPDATE Modify rows of data in a table.
 - DELETE Delete rows of data from a table.
 - REFERENCES Reference columns of named table in integrity constraints.
 - USAGE Use domains, collations, character sets, and translations.



GRANT

- Owner of table must grant other users the necessary privileges using GRANT statement.

```
GRANT {PrivilegeList | ALL PRIVILEGES}  
ON ObjectName  
TO {AuthorizationIdList | PUBLIC}  
[WITH GRANT OPTION]
```

- PrivilegeList consists of one or more of above privileges separated by commas.
- ALL PRIVILEGES grants all privileges to a user.

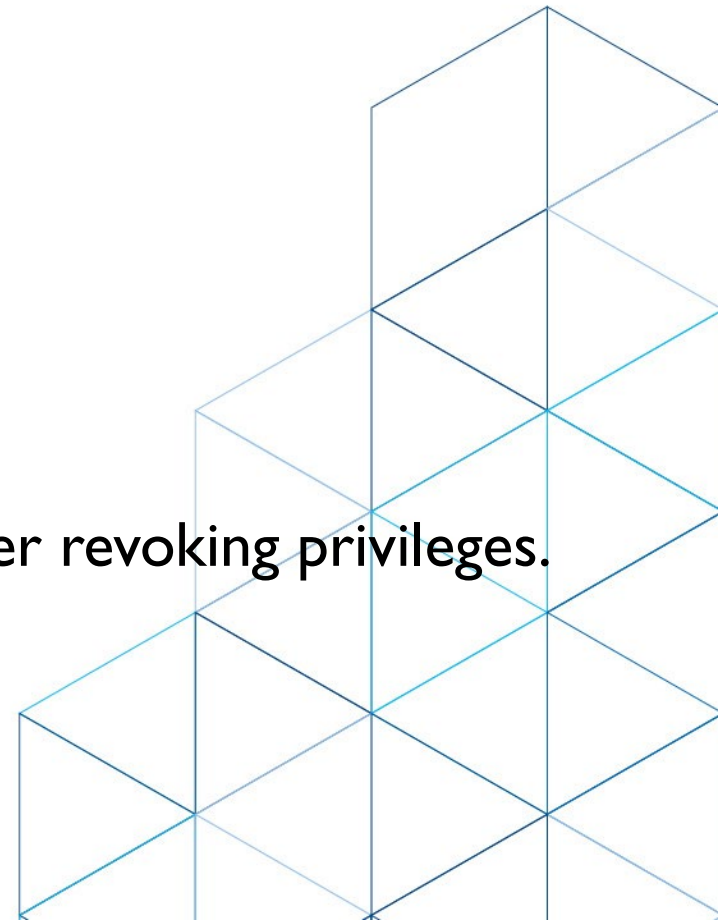


REVOKE

- REVOKE takes away privileges granted with GRANT.

```
REVOKE [GRANT OPTION FOR]
      {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
[RESTRICT | CASCADE]
```

- ALL PRIVILEGES refers to all privileges granted to a user by user revoking privileges.



Control Statements

- Conditional IF statement
- Conditional CASE statement
- Iteration statement (LOOP)
- Iteration statement (WHILE and REPEAT)
- Iteration statement (FOR)

Example of Exception Handling in PL/SQL

```
DECLARE
    vpCount    NUMBER;
    vStaffNo PropertyForRent.staffNo%TYPE := 'SG14';
    -- define an exception for the enterprise constraint that prevents a member of staff
    -- managing more than 100 properties
    e_too_many_properties EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_too_many_properties, -20000);
BEGIN
    SELECT COUNT(*) INTO vpCount
    FROM PropertyForRent
    WHERE staffNo = vStaffNo;
    IF vpCount = 100
    -- raise an exception for the general constraint
    RAISE e_too_many_properties;
    END IF;
    UPDATE PropertyForRent SET staffNo = vStaffNo WHERE propertyNo = 'PG4';
EXCEPTION
    -- handle the exception for the general constraint
    WHEN e_too_many_properties THEN
        dbms_output.put_line('Member of staff ' || staffNo || ' already managing 100 properties');
END;
```

Triggers

- Defines an action that the database should take when some event occurs in the application
- Based on Event-Condition-Action (ECA) model
- Types
 - Row-level
 - Statement-level
- Event: INSERT, UPDATE or DELETE
- Timing: BEFORE, AFTER or INSTEAD OF

```
CREATE TRIGGER TriggerName
  BEFORE | AFTER | INSTEAD OF
  INSERT | DELETE | UPDATE [OF TriggerColumnList]
  ON TableName
  [REFERENCING {OLD | NEW} AS {OldName | NewName}]
  [FOR EACH {ROW | STATEMENT}]
  [WHEN Condition]
  <trigger action>
```

```
CREATE TRIGGER StaffAfterInsert
  AFTER INSERT ON Staff
  REFERENCING NEW AS new
  FOR EACH ROW
  BEGIN
    INSERT INTO StaffAudit VALUES (:new.staffNo,
    :new.fName, :new.lName, :new.position,
    :new.sex, :new.DOB, :new.salary,
    :new.branchNo);
  END;
```

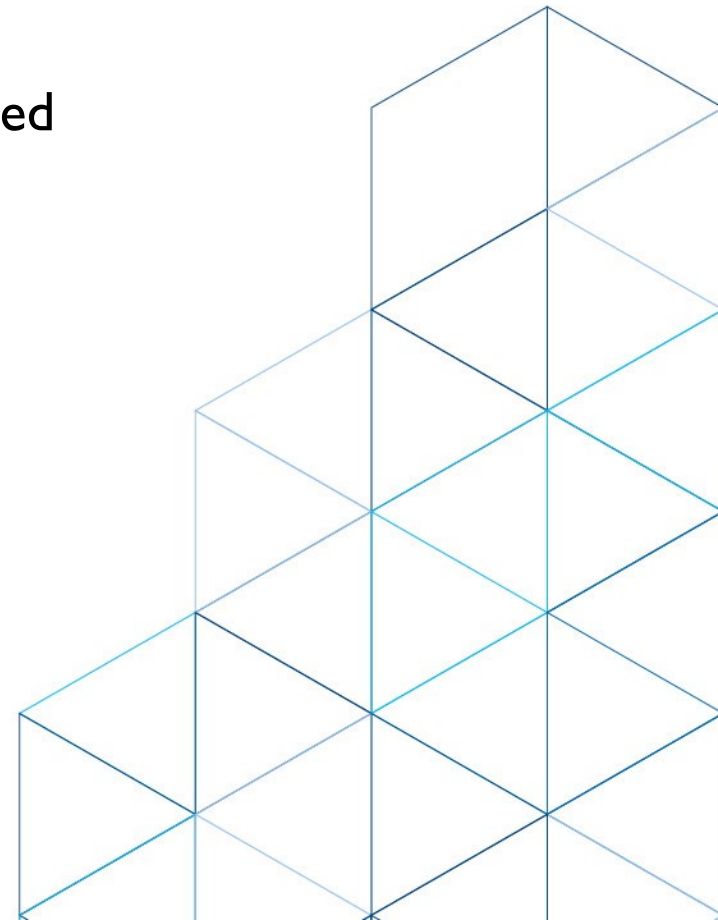
Triggers

- **Advantages**

- Elimination of redundant code
- Simplifying modifications
- Increased security
- Improved integrity
- Improved processing power
- Good fit with client-server architecture

- **Disadvantages**

- Performance overhead
- Cascading effects
- Cannot be scheduled
- Less portable



Comments or Questions?



Data Manipulation Language (DML)



Data Manipulation Language (DML)

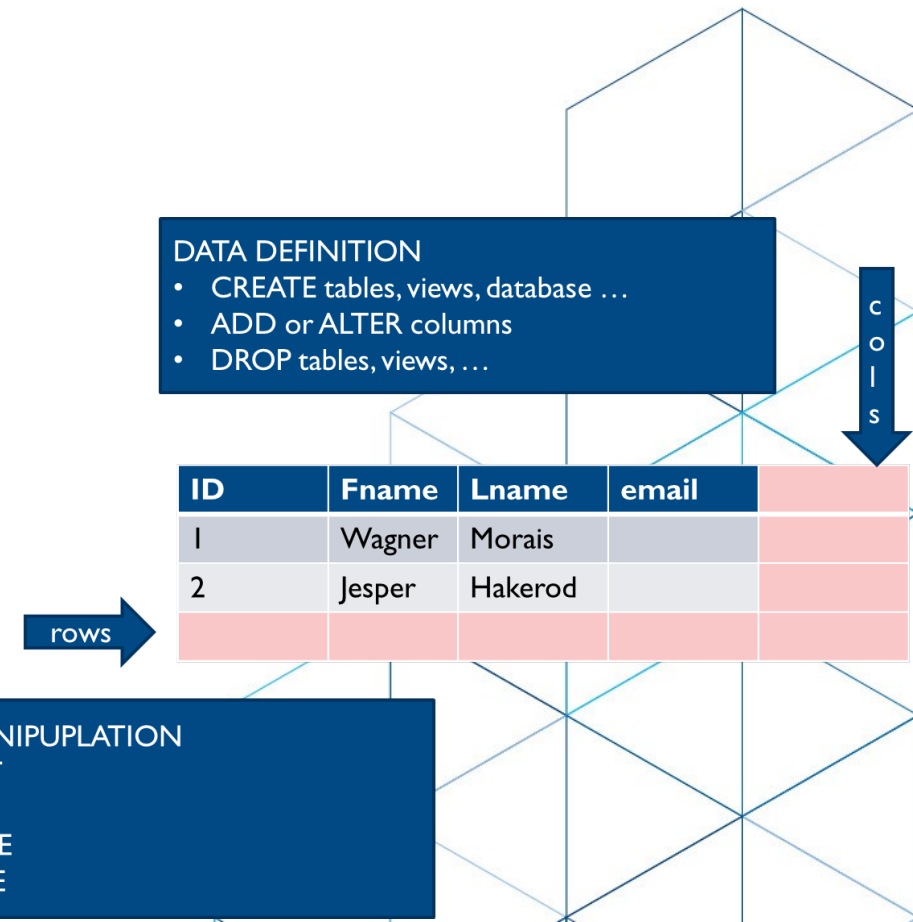
- How to retrieve data from database using SELECT and:
 - Use compound WHERE conditions.
 - Sort query results using ORDER BY.
 - Use aggregate functions.
 - Group data using GROUP BY and HAVING.
 - Use subqueries.
 - Join tables together.
 - Perform set operations (UNION, INTERSECT, EXCEPT).
- How to update database using INSERT, UPDATE, and DELETE.



Data Manipulation Language (DML)

- DML statements are used for querying, inserting, modifying, and deleting data.
 - SELECT – to query data in a table
 - INSERT – to insert data into a table
 - UPDATE – to update data in a table
 - DELETE – to delete data from a table

```
SELECT cu.customer_id AS id,  
       (((cu.first_name)::text || ' '::text) || (cu.last_name)::text) AS name,  
       a.address,  
       a.postal_code AS "zip code",  
       a.phone,  
       city.city,  
       country.country,  
       CASE  
         WHEN cu.activebool THEN 'active'::text  
         ELSE ''::text  
       END AS notes,  
       cu.store_id AS sid  
FROM (((customer cu  
      JOIN address a ON ((cu.address_id = a.address_id)))  
      JOIN city ON ((a.city_id = city.city_id)))  
      JOIN country ON ((city.country_id = country.country_id)));
```



SELECT Statement

- **Must have**
 - SELECT
 - Specifies which columns are to appear in output.
 - FROM
 - Specifies table(s) to be used.
- **Optional**
 - WHERE
 - Filters rows.
 - GROUP BY
 - Forms groups of rows with same column value.
 - HAVING
 - Filters groups subject to some condition.
 - ORDER BY
 - Specifies the order of the output

```
SELECT *  
FROM weather  
WHERE city = 'San Francisco'  
AND prcp > 0.0;
```

```
SELECT DISTINCT city  
FROM weather ORDER BY city;
```

```
SELECT *  
FROM weather, cities  
WHERE city = name;
```

Simple Queries - SELECT Statement

```
SELECT [DISTINCT | ALL]
{* | [columnExprn [AS newName]] [, ...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList]
```

SQL keyword that indicates
the type of query (in this case a
query to retrieve data)

SELECT

ISBN10, Title

Fields to retrieve

SQL keyword for specifying
the tables

FROM Books

Table to retrieve from

SELECT * **FROM** Books

Wildcard to select all fields

*Note: While the wildcard is convenient,
especially when testing, for production code it
is usually avoided; instead of selecting every
field, you should select just the fields you need.*



Simple Query Example

- All columns, all rows

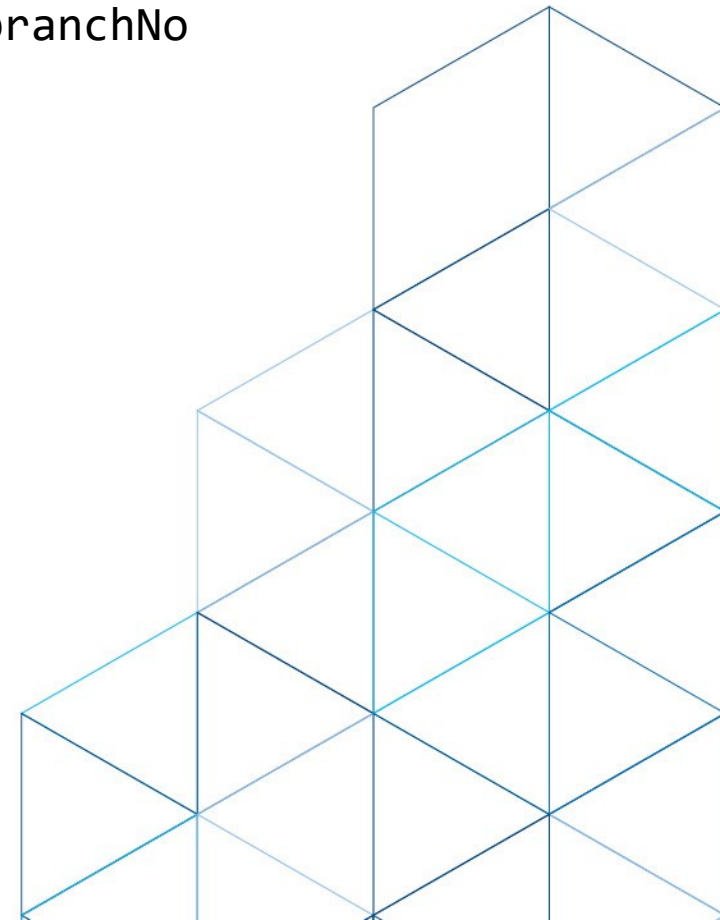
- List full details of all Staff

```
SELECT staffNo, fName, IName, position, sex, DOB, salary, branchNo  
FROM Staff;
```

- Can use * as an abbreviation for 'all columns':

```
SELECT *  
FROM Staff;
```

staffNo	fName	IName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005



Simple Query Example II

- Specific columns, all rows

- List of salaries for all staff, showing only the staff number, the first and last names, and the salary details.

```
SELECT staffNo, fName, IName, position  
FROM Staff;
```

- Calculated Fields

- Produce a list of monthly salaries for all staff, showing the staff number, the first and last names, and the salary details.

```
SELECT staffNo, fName, IName, position, salary/12  
FROM Staff;
```

staffNo	fName	IName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

staffNo	fName	IName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

staffNo	fName	IName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

Simple Query Example III

- Use of DISTINCT

- List the property numbers of all properties that have been viewed.

```
SELECT propertyNo  
FROM Viewing;
```

- Use DISTINCT to eliminate duplicates

```
SELECT DISTINCT propertyNo  
FROM Viewing;
```

propertyNo

PA14

PG4

PG36

propertyNo

PA14

PG4

PG4

PA14

PG36

SELECT - WHERE Clause

- Often we are not interested in retrieving all the records in a table but only a subset of the records.

```
SELECT isbn10, title FROM books  
WHERE copyrightYear > 2010
```

SQL keyword that indicates
to return only those records
whose data matches the
criteria expression

Expressions take form:
field *operator* value

```
SELECT isbn10, title FROM books  
WHERE category = 'Math' AND copyrightYear = 2014
```

Comparisons with strings require string
literals (single or double quote)



Simple Query Example IV

- Comparison Search Condition

- List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000;
```

- Compound Comparison Search Condition

- List addresses of all branch offices in London or Glasgow.

```
SELECT *
FROM Branch
WHERE city = 'London' OR
city = 'Glasgow';
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

Simple Query Example V

- Set Membership

- List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

- Could have expressed this as

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position='Manager' OR position='Supervisor';
```

- But, IN is more efficient when set contains many values.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Simple Query Example VI

- Pattern Matching
 - SQL has two special pattern matching symbols:
 - %: sequence of zero or more characters;
 - _ (underscore): any single character.
 - Find all owners with the string 'Glasgow' in their address.
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE '%Glasgow%';

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025



SELECT - Ordering results

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1, column2, ... ASC|DESC;
```

```
SELECT staffNo, fName, lName, salary  
FROM Staff  
ORDER BY salary DESC;
```

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

```
select iSbN10, title  
FROM BOOKS  
ORDER BY title
```

SQL keyword
to indicate
sort order

Field to
sort on

Note: SQL doesn't care if a command is on a single line or multiple lines, nor does it care about the case of keywords or table and field names. Line breaks and keyword capitalization are often used to aid in readability.

```
SELECT ISBN10, Title FROM Books  
ORDER BY CopyrightYear DESC, Title ASC
```

Keywords indicating that sorting should be in descending or ascending order (which is the default)

Several sort orders can be specified: in this case the data is sorted first on year, then on title.

SELECT - Aggregates

- ISO standard defines five aggregate functions:
 - COUNT returns number of values in specified column.
 - SUM returns sum of values in specified column.
 - AVG returns average of values in specified column.
 - MIN returns smallest value in specified column.
 - MAX returns largest value in specified column.

- Each operates on a single column of a table and returns a single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.
- COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.
- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.
- Aggregate functions can be used only in SELECT list and in HAVING clause.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Find number of Managers and sum of their salaries.

```
SELECT COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff
WHERE position = 'Manager';
```

myCount	mySum
2	54000.00

SELECT – Grouping

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

- Use GROUP BY clause to get sub-totals.
- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.

- Find number of staff in each branch and their total salaries.

```
SELECT branchNo, COUNT(staffNo) AS myCount, SUM(salary) AS mySum  
FROM Staff
```

```
GROUP BY branchNo
```

```
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

SELECT – Having

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

- HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.
- For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT branchNo, COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Subqueries

- Some SQL statements can have a **SELECT** embedded within them.
- A subselect can be used in **WHERE** and **HAVING** clauses of an outer **SELECT**, where it is called a subquery or nested query.
- Subselects may also appear in **INSERT**, **UPDATE**, and **DELETE** statements.

- List staff who work in branch at '163 Main St'.

```
SELECT staffNo, fName, lName, position  
FROM Staff
```

```
WHERE branchNo = ( SELECT branchNo  
                   FROM Branch  
                   WHERE street = '163 Main St');
```

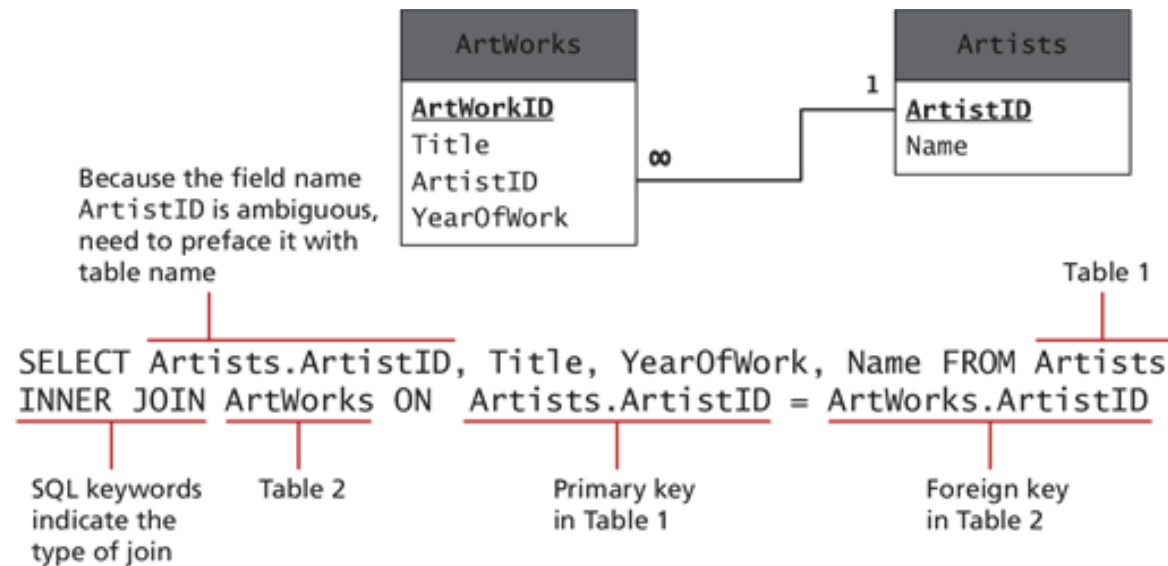
staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005



staffNo	fName	lName	position
SG37	Ann	Beech	Assistant
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

SELECT - Advanced (but essential) techniques

- Often we want to retrieve data from multiple tables. While we could do one query, and then do a second query using the data from the 1st query, that would be inefficient.
- Instead we can use the JOIN clause.



SELECT - Advanced (but essential) techniques

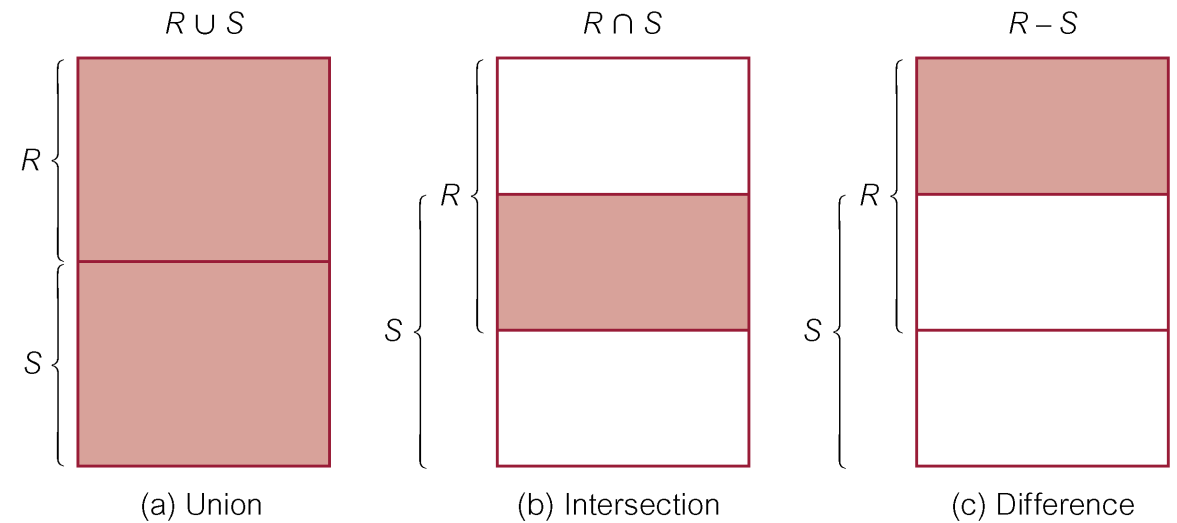
- Or, include more than one table in FROM clause.
- Use comma as separator and typically include WHERE clause to specify join column(s).
- Also possible to use an alias for a table named in FROM clause.
- Alias is separated from table name with a space.
- Alias can be used to qualify column names when there is ambiguity.
- List names of all clients who have viewed a property along with any comment supplied.

```
SELECT c.clientNo, fName, lName, propertyNo, comment
FROM Client c, Viewing v
WHERE c.clientNo = v.clientNo;
```

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

Union, Intersect, and Except (Difference)

```
(SELECT city  
FROM Branch  
WHERE city IS NOT NULL)  
UNION  
(SELECT city  
FROM PropertyForRent  
WHERE city IS NOT NULL);
```



INSERT - Adding data into a table

```
INSERT INTO TableName [ (columnList) ]  
VALUES (dataValueList)
```

- columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.

SQL keywords for inserting
(adding) a new record

Table name

Fields that will
receive the data values

```
INSERT INTO ArtWorks (Title, YearOfWork, ArtistID)  
VALUES ('Night Watch', 1642, 105)
```

Values to be inserted. Note that string values
must be within quotes (single or double).

*Note: Primary key fields are
often set to AUTO_INCREMENT,
which means the DBMS will set
it to a unique value when a new
record is inserted.*

INSERT - Adding data into a table

- Insert using default

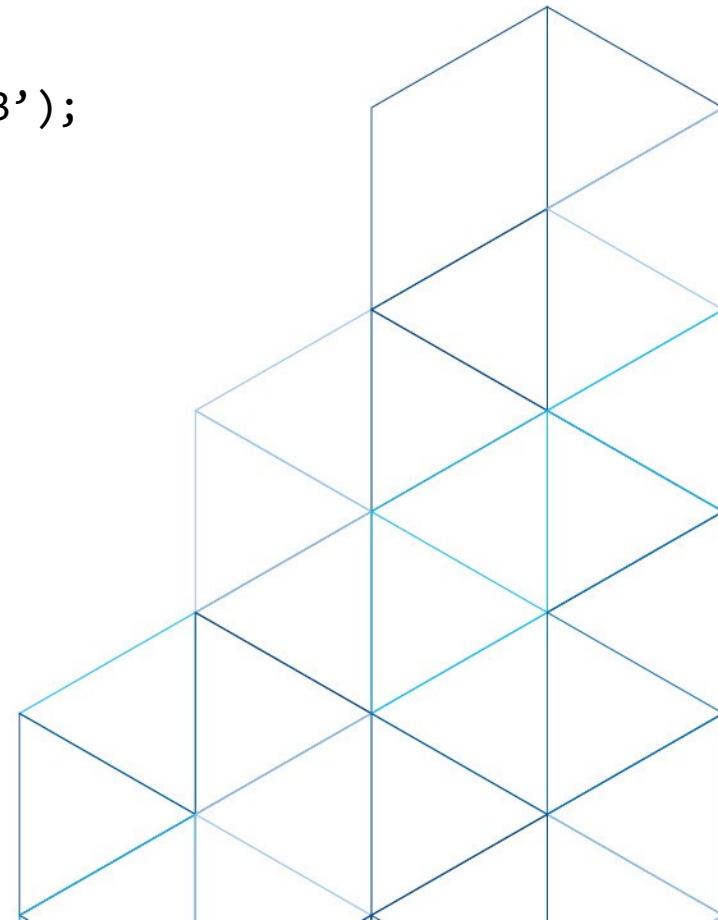
```
INSERT INTO Staff (staffNo, fName, lName, position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', 8100, 'B003');
```

– Or

```
INSERT INTO Staff
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL, NULL, 8100, 'B003');
```

- Insert using select

```
INSERT INTO StaffPropCount
(SELECT s.staffNo, fName, lName, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.staffNo, fName, lName)
UNION
(SELECT staffNo, fName, lName, 0
FROM Staff
WHERE staffNo NOT IN
(SELECT DISTINCT staffNo
FROM PropertyForRent));
```



UPDATE - Modify data in the table

UPDATE TableName

SET columnName1 = dataValue1 [, columnName2 = dataValue2...]

[WHERE searchCondition]

- TableName can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.
- WHERE clause is optional:
 - if omitted, named columns are updated for all rows in table;
 - if specified, only those rows that satisfy searchCondition are updated.
- New dataValue(s) must be compatible with data type for corresponding column.

```
UPDATE ArtWorks
```

```
SET Title='Night Watch', YearOfWork=1642, ArtistID=105
```

```
WHERE ArtWorkID=54
```

It is essential to specify which record to update, otherwise it will update all the records!

Specify the values for each updated field.
Note: Primary key fields that are AUTO_INCREMENT cannot have their values updated.

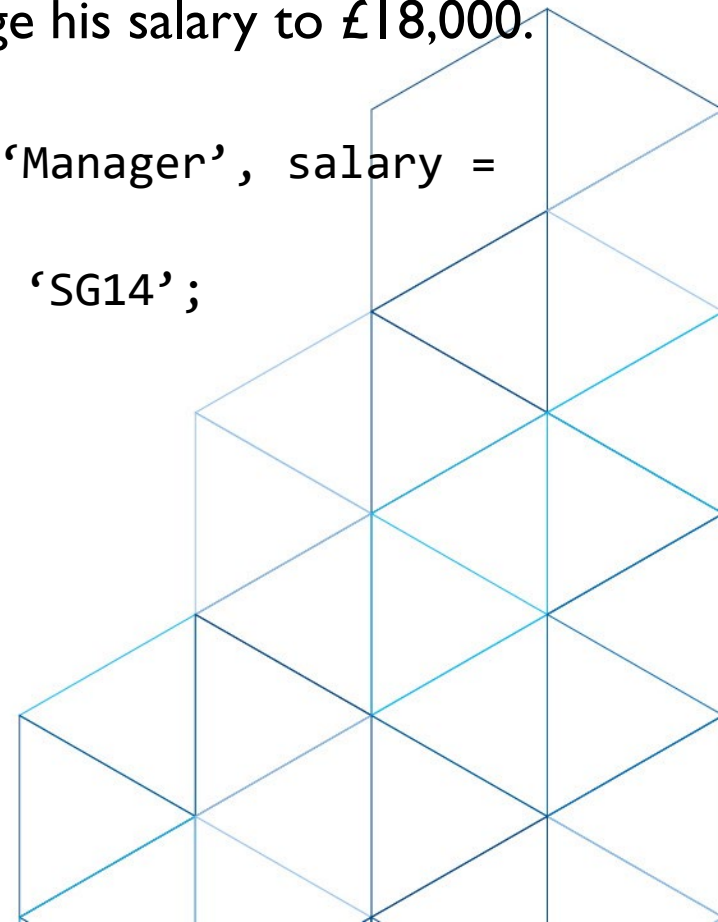
UPDATE examples

- All rows
 - Give all staff a 3% pay increase.

```
UPDATE Staff  
SET salary = salary*1.03;
```
 - Give all Managers a 5% pay increase.

```
UPDATE Staff  
SET salary = salary*1.05  
WHERE position = 'Manager';
```
- Multiple columns, single row
 - Promote David Ford (staffNo='SG14') to Manager and change his salary to £18,000.

```
UPDATE Staff  
SET position = 'Manager', salary = 18000  
WHERE staffNo = 'SG14';
```



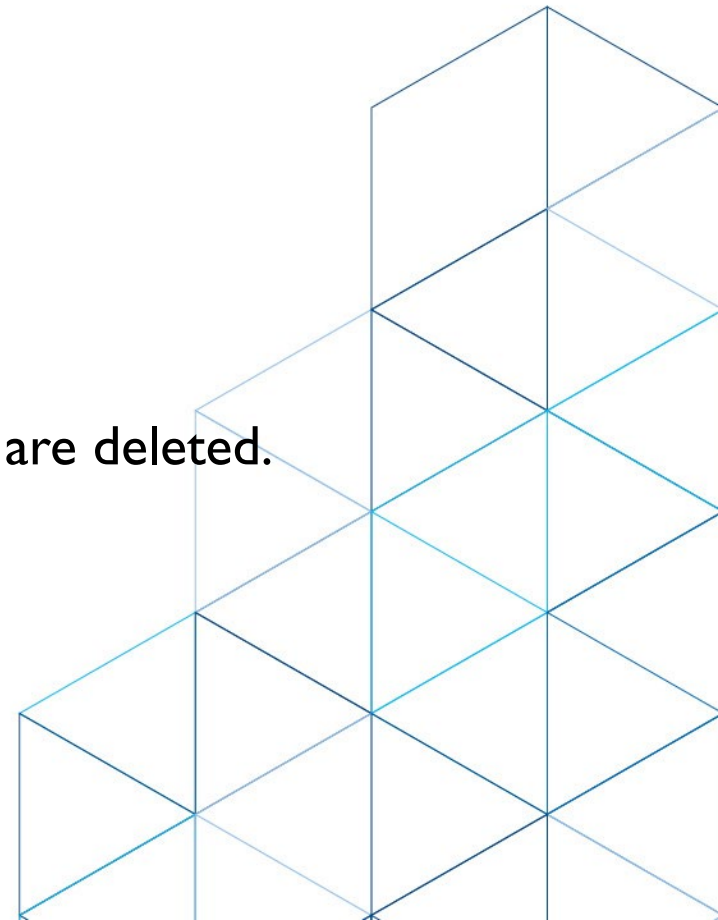
DELETE

```
DELETE FROM TableName  
[WHERE searchCondition]
```

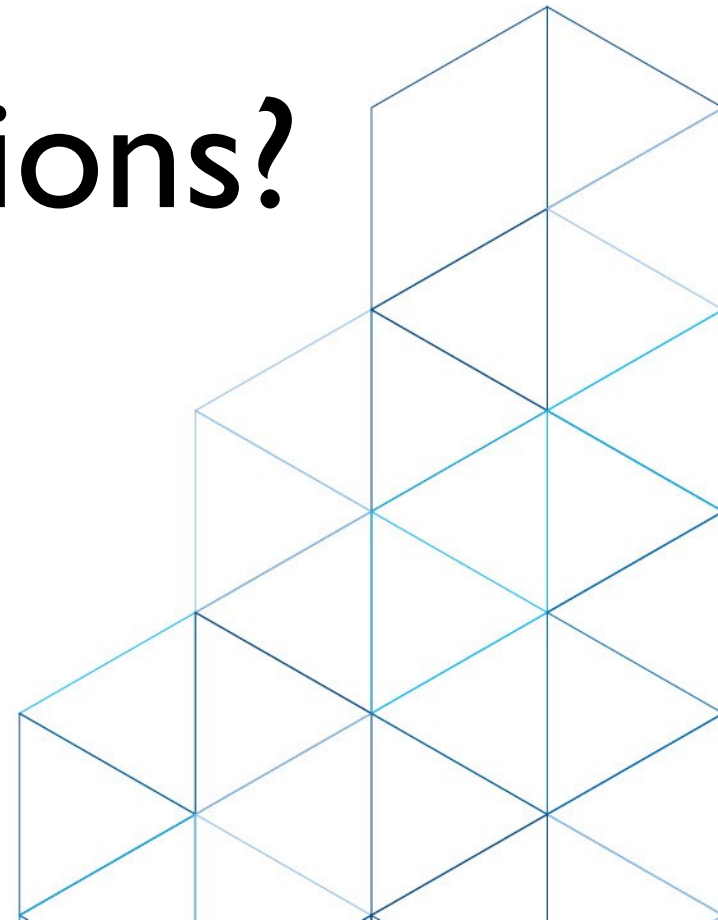
- TableName can be name of a base table or an updatable view.
 - searchCondition is optional;
 - if omitted, all rows are deleted from table. This does not delete table.
- ```
DELETE FROM ArtWorks
```
- will delete all records from the ArtWorks table.
  - if search\_condition is specified, only those rows that satisfy condition are deleted.

```
DELETE FROM ArtWorks
WHERE ArtWorkID=54
```

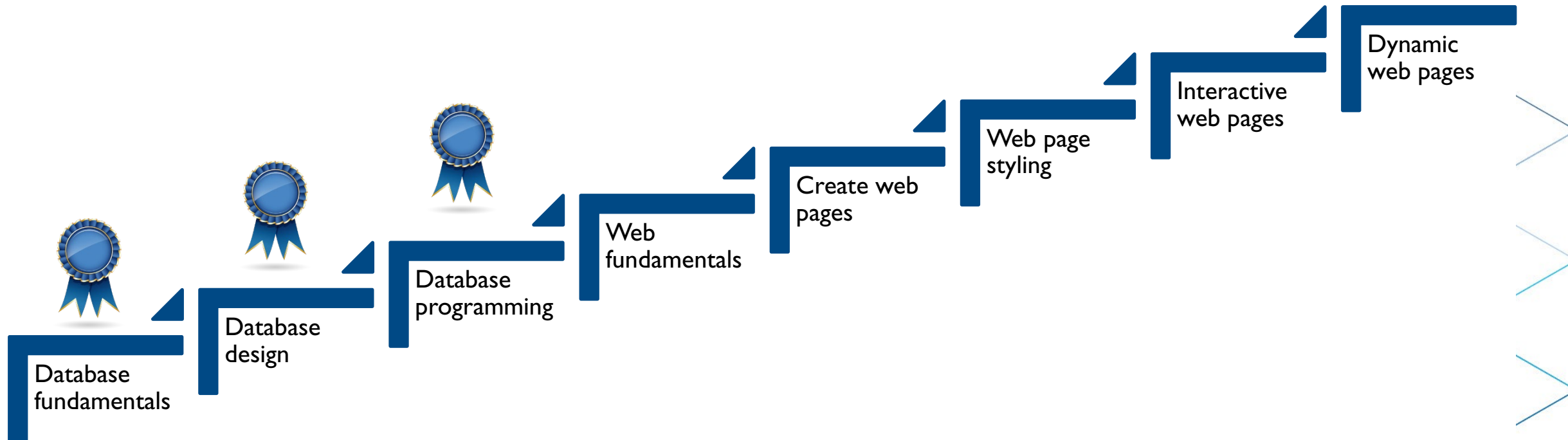
It is essential to specify which record to delete, otherwise it will delete all the records!



# Comments or Questions?



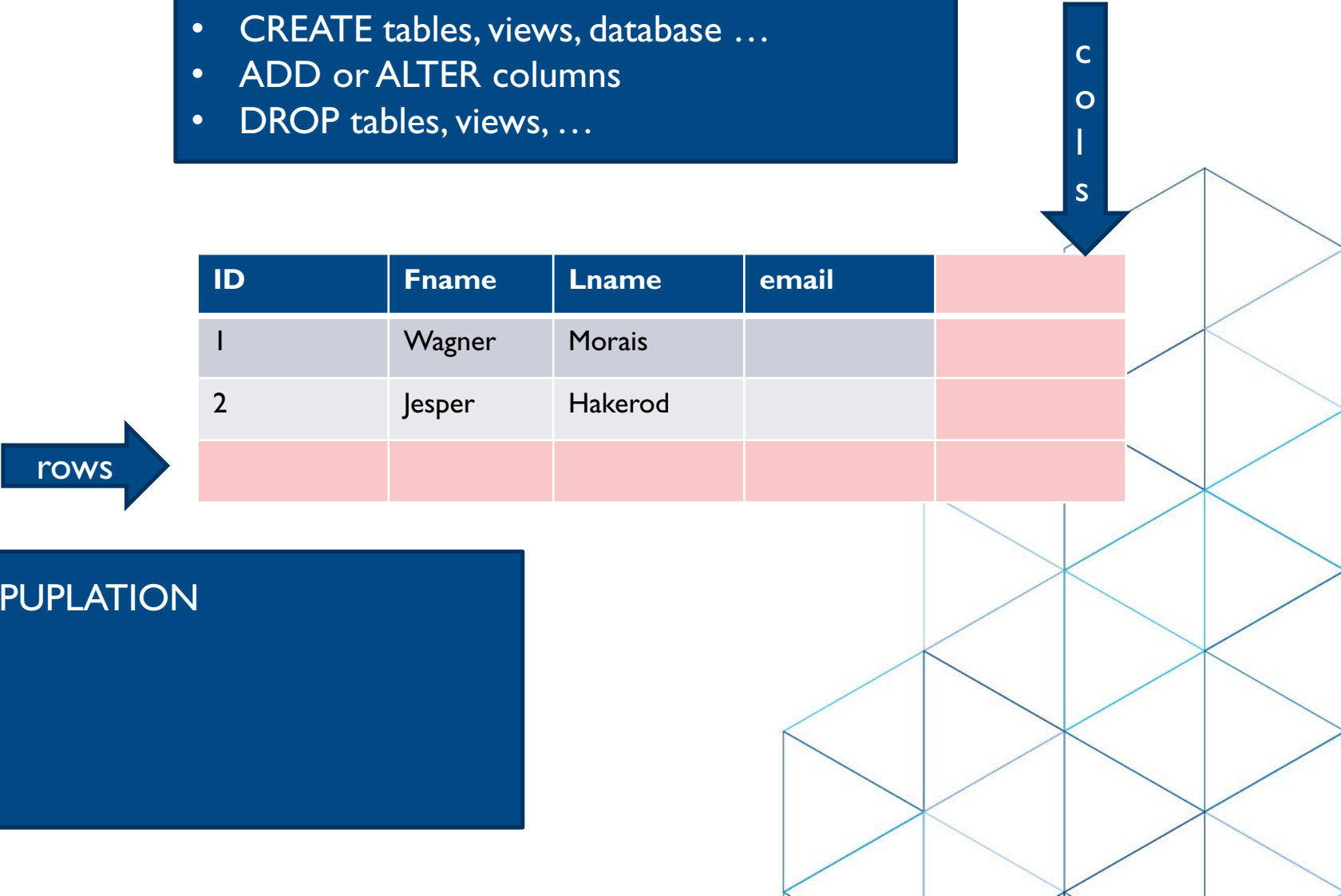
# Congratulations



# DDD and DML

## DATA DEFINITION

- CREATE tables, views, database ...
- ADD or ALTER columns
- DROP tables, views, ...



| ID | Fname  | Lname   | email |  |
|----|--------|---------|-------|--|
| 1  | Wagner | Morais  |       |  |
| 2  | Jesper | Hakerod |       |  |
|    |        |         |       |  |

## DATA MANIPULATION

- SELECT
- INSERT
- UPDATE
- DELETE