What is a database?

* A structured set of data held in a computer
* There are numerous types of databases:

. Relational Databases / RDBMS: emerged in the 70s, store data according to a schema that allows data to be displayed as tables with rows & columns.

. No SQL/non-relational: alternate to RDBMS

Types:

* Key-value stores. i.e Amazon DynamoDB
* Wide column stores i.e Cassandra
* Document stores i.e. mongoDB
* Graph databases i.e. Neo4T

SQL – structured Query Language

* How we interact with databases
* Many similar dialects, based on the database provider.
* Database schema – the organization of data as a blueprint of how the DB is constructed.
* ERD – Entity Relationship Diagram

. Model which describes interrelated data, it shows the composition of the entities & the relationships between them.

* Entity – any object in the system that we want to model & store data about, represented as a table in an RDBMS.
* Fields – properties of entities – represented as columns.

In a properly normalized DB, the integrity of the relationships between tables is reliant on 2 keys (special columns): PK & FK.

* Candidate key = column or set of columns which can uniquely identify a row.
* Primary key (PK) – candidate key which is not null & unique, and will not change.
* Foreign key – reference to another field, usually the PK of another table.
* A **composite key** is a combination of two or more columns in a table that can be used to uniquely identify each row in the table.
* An *orphaned record* is a record whose [foreign key](https://database.guide/what-is-a-foreign-key/) value references a non-existent [primary key](https://database.guide/what-is-a-primary-key/) value.

MULTIPLICITY: Relationship between tables

1 : 1 i.e Person – phone$

Person – SSN

1 : N i.e Person – accounts

Cat – Cat breeds

N : N i.e Student – courses

User follows – user following.

DB Normalization is the process of organizing data to increase efficiency, integrity, and reduce redundancy.

1NF “the key”; every table has a PK, columns broken down into atomic values, no repeated columns.

2NF “the whole key”:

* Has to be 1NF
* No partial dependencies

3NF “Nothing but the key”:

* 2NF
* No transitive dependencies.

**DCL:** Data Control Language: give access credits to users: GRANT, REVOKE

**DDL**: Data Definition Language

* Defines schema of DB: CREATE, DROP, TRUNCATE, ALTER

CREATE makes new DB objects i.e. tables, sequences, stored Procedure Functions.

Drop: remove any object, sequences, remove schema.

Truncate: remove all rows from a table keep schema.

Alter is to alter schema, is used to add, delete, or modify columns in an existing table.

**DML** = Data Manipulation Language “CRUD”

INSERT, SELECT, UPDATE, DELETE

**DQL** = Data Query Language: Select

**TCL**: Transaction Control Language

Commit, Rollback, savepoint.

Constrains: restrictions on data input / entity behavior.

CASCADE: do the following that.

Column level constraints

* Not null
* Unique
* Primary key – implies NN, U.
* Foreign key
* Default
* Check

SET OPERATORS: used to compare result sets with same # of, and type of columns. Result sets are rows returned from query.

* + UNION:
    - Return all distinct values between tables
    - A+ B -AB
  + UNION ALL
    - Includes duplicates
    - A+B
  + INTERSECT
    - Returns the rows that tables share in common
    - AB
  + MINUS
    - Rows that are in A that are not in B

**Join and union difference**: The columns of joining tables may be different in JOIN but in UNION the number of columns and order of columns of all queries must be same.

2.) The UNION puts rows from queries after each other (puts vertically) but JOIN puts the column from queries after each other (puts horizontally), i.e. it makes a cartesian product.

-->Scalar functions-functions that operate on single values. i.e upper() lower() lenght()

-->Aggregate functions - functions that operate on multiple rows of a column ie max() min() count() sum()

* **WHERE vs HAVING**
  + Use having when following GROUP BY
  + GROUP BY: Groups rows defined by a specific set of columns for using aggregate functions on the rows in those groups.
  + ORDER BY: Sorts the returned query by a specific set of columns
  + DISTINCT
  + LIKE

**Referential Integrity**

Ensuring that relationships between entities remain consistent. Ie forbidding the deletion of columns referenced by fk’s, references to fields that do not exist, etc.

**Domain integrity**

Ensuring that fields are of the right type + size.

**Sequences**

* Good for PK management
* Variable objects stored in DB
* Create sequence[name]

Start with[value]

Increment by [value]

Min value[value]

Max value[val]

Cache [amt]: how many values will be stored in memory for faster access.

|  |
| --- |
| --SUBQUERIES |
|  | /\*Queries nested within the where clause of another query to further narrow the result set. |
|  | non-correlated - inner query can execute independently from the outer |
|  | correlated - internal query cannot execute independently |
|  | \*/ |
|  | SELECT \* FROM GENRE; |
|  | SELECT GENREID FROM GENRE WHERE NAME = 'Drama'; |
|  | SELECT \* FROM TRACK WHERE GENREID = 20; |
|  |  |
|  | --here, nested query returns only one row |
|  | --SELECT from trak where my genre is Drama |
|  | SELECT \* FROM TRACK WHERE GENREID = (SELECT GENREID FROM GENRE WHERE NAME = 'Drama'); |
|  |  |
|  | SELECT \* FROM track WHERE genreid IN |
|  | (SELECT genreid FROM genre WHERE NAME LIKE 'B%'); |

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |
| --- |
|  |
|  | JOINS: join clause is used to join columns based on a relationship between them, |
|  | typically used to bring together columns from different tables based on a foreign key relationship. |
|  |  |
|  |  |
|  | -- Join without using the join keyword |
|  | select album.title, artist.name |
|  | from album, artist where album.artistid = artist.artistid; |
|  |  |
|  | --INNER JOIN AKA JOIN |
|  | select art.name as ARTIST, al.title AS "ALBUM TITLE", t.name as SONG |
|  | from album al |
|  | join artist art on al.artistid = art.ARTISTID |
|  | join track t on t.ALBUMID = al.ALBUMID |
|  | order by art.name;  Self-join: 1 table  select e1.lastname as EMPLOYEE, e2.lastname as MANAGER  from employee e1  join employee e2  on e1.reportsto = e2.employeeid;   * **(INNER) JOIN**: Returns records that have matching values in both tables * **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table * **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table * **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table |

A trigger is a special type of stored procedure that automatically executes when an event occurs in the database server.

DML triggers execute when a user tries to modify data through a DML event. We will use them BEFORE each INSERT on table

to get the next value of our sequence and assign it to the PK value.

create or replace trigger BN\_BOOK\_TRIG -- declare and name trigger

before insert on BN\_BOOK -- when will trigger execute

for each row -- necessary to change value in a table

begin

-- series of SQL statements to operate when event happens

-- Incrementing book seq and assigning it to a new book\_id(book's pk) value

select BN\_BOOK\_SEQ.nextval into : NEW.BOOK\_ID from dual;

end;

/

PL/SQL – Procedure Language extension of SQL.

* Blocks of code used to define functionality compiled & stored in the database.

Index: - DB entity that can be applied to a column to enforce a physical in memory ordering of its rows.

* Speeds up searches, but slows down inserts/updates. Use sparingly.
* Unique, FK, PK constraints automatically create “backing” indexes. Cannot add another.
* Create index [name] on [table](col)ASC/DESC;

**Cursors**:

* Oracle creates a “context area” with all information necessary for processing an SQL statement.
* A cursor is a pointer to a context area.
* The “active set” is the rows returned by a query & is held by the cursor.
* Implicit cursor – created by Oracle whenever a DML statement is ran & no explicit cursor on the set exists. No control over these.
* Explicit cursor – programmer defined. Used to obtain control over the context area.

1. DECLARE cursor to initialize memory
2. OPEN – allocate memory
3. FETCH – retrieve data
4. CLOSE – release memory

**Functions**:

* Executable block of code which must return a value and may have many parameters.
* DQL statements only
* Invoke using ()

**Stored Procedures**

* Executable block of code without O+|N|OUT parameters & no return value
* Full DML & TCL capabilities.

create or replace procedure get\_all\_books

(book\_cursor OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN book\_cursor FOR select \* from bn\_book;

end;

/

**TRANSACTIONS**

* Units of work done on a database
* Many include many DML command

**Properties of a TX (ACID)**

ATOMICITY – “all or nothing” – either all operations of the tx execute successfully, or no commit is made.

Consistency – DB is in a valid state according to existing structure & constraints after a commit.

Isolation – The system state during concurrent transactions is the same as if the tx’s were sequential.

Durability – All commits are final & cannot be rolled back – even in case of system failure.

**ISOLATION**

**Dirty Read**

Session 1 – Begins TX, modifies data.

S2 – begins TX with S1’s uncommitted data.

S1 – rolls back data.

S2 - still working with S1’s old & invalid data.

**Non-repeatable read**:

* S1 – begin TX & retrieve data
* S2 – update data including data being viewed by S1 commits.
* S1 – attempts to retrieve same data during same session but can’t.

**Phantom Read**:

* S1 – begin tx & retrieve data
* S2 – insert data matching same query s1 user retrieved.
* S1 – retrieves data with same query but “phantom” data has appeared.

|  |  |  |  |
| --- | --- | --- | --- |
| Isolation levels | Dirty Read | Non-repeatable Read | Phantom Read |
| Read uncommitted | Y | Y | Y |
| Read committed |  | Y | Y |
| Repeatable read |  |  | Y |
| Serializable |  |  |  |

Statement: Use for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

\* - takes an SQL statement as a string, executes it, and returns the result.

\* - allow SQL injection. These are not ideal to use. Definitly do not use for any SQL.

\* command that uses a parameter/variable

\* SQL injection - common hacking technique. It is the insertion of code as input that affects your database

PreparedStatement: Use when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.

CallableStatement: Use when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

* **Important interfaces in the JDBC API**
  + Connection:
  + Statement:
  + PreparedStatement:
  + CallableStatement:
  + ResultSet:
* **executeUpdate() vs executeQuery()**
* **How, in Java, do i obtain the keys generated (via triggers, etc) in my database?**
  + getAutoGeneratedKeys()

create table employee

(first varchar(15) primary key,

last varchar(20),

age number(3),

address varchar(30),

city varchar(20),

state varchar(20));

**ALTER TABLE Student ADD (AGE number(3),COURSE varchar(40));**

**ALTER TABLE Student MODIFY COURSE varchar(20);**

**ALTER TABLE Student DROP COLUMN COURSE;**

**DROP TABLE Student;**

The TRUNCATE TABLE mytable statement is logically (though not physically) equivalent to the DELETE

**TRUNCATE TABLE table\_name;**

**ALTER TABLE Student RENAME COLUMN NAME TO FIRST\_NAME;**

select first, last, city

from empinfo

where first LIKE 'Er%';

SELECT COUNT (\*) FROM Sales WHERE CustomerName = 'Smith'

SELECT \* FROM Users ORDER BY FirstName

SELECT DISTINCT City FROM Users

SELECT CustomerName, SUM(OrderPrice) FROM Sales GROUP BY CustomerName HAVING SUM(OrderPrice) > 1200

insert into employee  
 (first, last, age, address, city, state)  
 values ('Luke', 'Duke', 45, '2130 Boars Nest',   
 'Hazard Co', 'Georgia');

update phone\_book  
 set last\_name = 'Smith', prefix=555, suffix=9292  
 where last\_name = 'Jones';

delete from employee  
 where lastname = 'May';

SELECT Manufacturer, ManufacturerWebsite, ManufacturerEmail, AVG(Price) AS AvgPrice FROM Manufacturer JOIN Product ON Manufacturer.ManufacturerID = Product.ManufacturerID GROUP BY Manufacturer, ManufacturerWebsite, ManufacturerEmail

Inner join: a matching row

Outer join: everything

SELECT Column1, Column2 FROM Table1 UNION SELECT Column1, Column2 FROM Table2