SQL

COSC 516 – Cloud Databases

SQL Overview

Structured Query Language or SQL is the standard database query language.

- It first became an official standard in 1986 as defined by the American National Standards Institute (ANSI).
- All major database vendors conform to the SQL standard with minor variations in syntax (different *dialects*).
- SQL consists of both a Data Definition Language (DDL) and a Data Manipulation Language (DML).

SQL is a ***declarative language*** (non-procedural). A SQL query specifies *what* to retrieve but not *how* to retrieve it.

- Basic SQL is not a complete programming language as it does not have control or iteration commands.
  - Procedural extensions: PL/SQL (Oracle), T-SQL (SQL Server)

SQL Basic Rules

1) There is a set of *reserved words* that cannot be used as names for database fields and tables.

- `SELECT`, `FROM`, `WHERE`, etc.

2) SQL is generally *case-insensitive*.

- Only exception is string constants. 'FRED' not the same as 'fred'.

3) SQL is *free-format* and white-space is ignored.

4) The semi-colon is often used as a statement terminator, although that is not always required.

5) Date and time constants have defined format:

  - Dates: 'YYYY-MM-DD' e.g. '1975-05-17'
  - Times: 'hh:mm:ss[.f] ' e.g. '15:00:00'
  - Timestamp: 'YYYY-MM-DD hh:mm:ss[.f]' e.g. '1975-05-17 15:00:00'

6) Two single quotes `''` are used to represent (escape) a single quote character in a character constant. e.g. `'Master''s'`.

SQL Identifiers

Identifiers are used to identify objects in the database such as tables, views, and columns.

- The identifier is the name of the database object.

An SQL identifier (name) must follow these rules:

- only contain upper or lower case characters, digits, and underscore ("_") character
- be no longer than 128 characters
  - DB vendors may impose stricter limits than this.
- must start with a letter (or underscore)
- cannot contain spaces
- Note: Quoted or ***delimited identifiers*** enclosed in double quotes allow support for spaces and other characters. E.g. `"select"`

SQL Data Types

In the relational model, each attribute has an associated *domain* of values.

In SQL, each column (attribute) has a ***data type*** that limits the values that it may store. The standard SQL data types are similar to their programming language equivalents.

The database will perform (implicit) data type conversion when necessary.

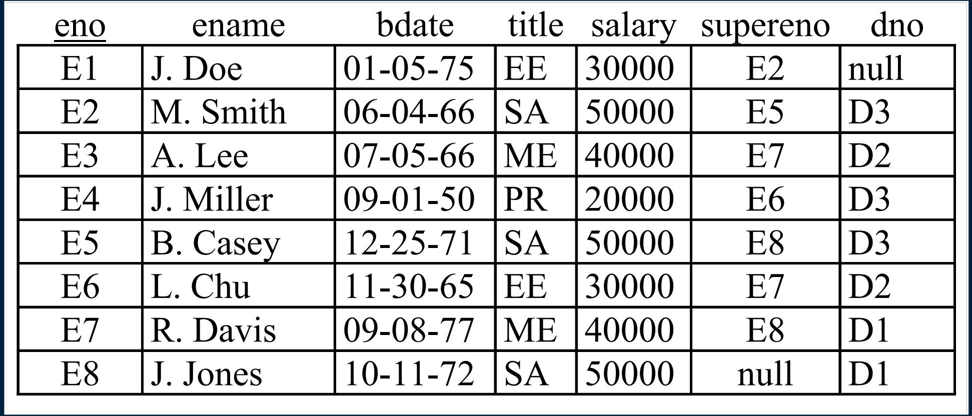
Explicit data type conversion using functions such as `CAST` and `CONVERT`.

SQL Data Types (2)

Data Type	Description
BOOLEAN	TRUE or FALSE
CHAR	Fixed length string (padded with blanks) e.g. CHAR(10)
VARCHAR	Variable length string e.g. VARCHAR(50)
BIT	Bit string e.g. BIT(4) can store '0101'
NUMERIC or DECIMAL	Exact numeric data type e.g. NUMERIC(7,2) has a precision (max. digits) of 7 and scale of 2 (# of decimals) e.g. 12345.67
INTEGER	Integer data only
SMALLINT	Smaller space than INTEGER
FLOAT or REAL	Approximate numeric data types. Precision dependent on implementation.
DOUBLE PRECISION	
DATE	Stores YEAR, MONTH, DAY
TIME	Stores HOUR, MINUTE, SECOND
DATETIME or TIMESTAMP	Stores date and time data.
INTERVAL	Time interval.
CHARACTER LARGE OBJECT	Stores a character array (e.g. for a document)
BINARY LARGE OBJECT	Stores a binary array (e.g. for a picture, movie)

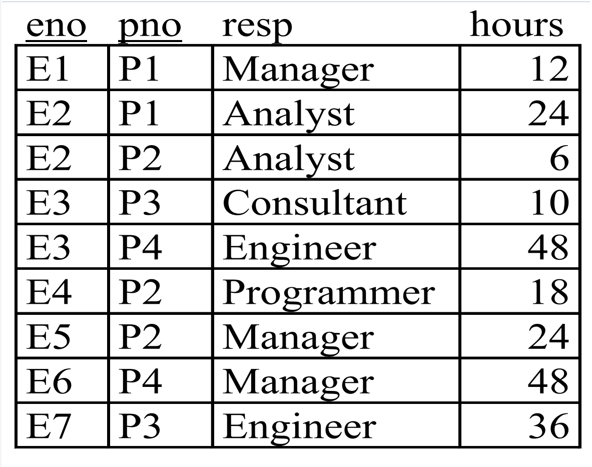
Example Database - WorksOn

emp Table



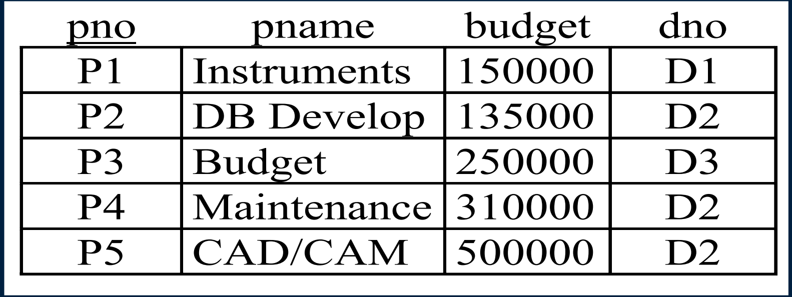
eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

workson Table



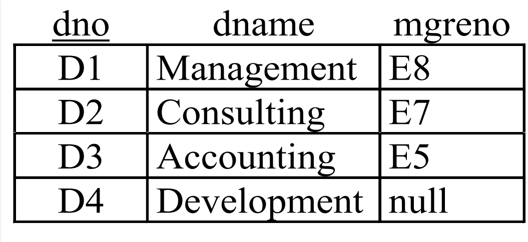
eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

proj Table



pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

dept Table



dno	dname	mgreno
D1	Management	E8
D2	Consulting	E7
D3	Accounting	E5
D4	Development	null

SQL CREATE TABLE

The **CREATE TABLE** command is used to create a table in the database. A table consists of a table name and a set of fields with their names and data types.

Example:

```
CREATE TABLE emp (
        eno      CHAR(5),
        ename    VARCHAR(30) NOT NULL,
        bdate    DATE,
        title    CHAR(2),
        salary   DECIMAL(9,2),
        supereno CHAR(5),
        dno      CHAR(5),
        PRIMARY KEY (eno)
)
```

field must always have a value

Data Types:

CHAR(5) – always 5 chars long

VARCHAR(30) – up to 30 chars long

DECIMAL(9,2) – e.g. 1234567.99

DATE – e.g. 1998/01/18

SQL Constraints

Constraints are specified in `CREATE` and `ALTER TABLE` statements.

Types of constraints:

- 1) **Required data** - To specify that a column must always have a data value (cannot be `NULL`) specify `NOT NULL` after the column definition.
  - e.g. `eno CHAR(5) NOT NULL`
  - If a field is `UNIQUE` or a `PRIMARY KEY`, `NOT NULL` is not necessary.
- 2) **Domain constraints** - Verify that the value of a column is in a given domain using `CHECK`.
  - e.g. `title CHAR(2) CHECK (title IN (NULL,'EE','SA','PR','ME'))`
  - Forces the title to be either `NULL` or one of 4 defined values.
  - Can also be performed using user-defined types (domains).

SQL Constraints - Entity Integrity

Entity Integrity constraint - The primary key of a table must contain a unique, non-null value for each row. The primary key is specified using the `PRIMARY KEY` clause.

- e.g. `PRIMARY KEY (eno)` (for emp relation)
- e.g. `PRIMARY KEY (eno,pno)` (for workson relation)
- It is also possible to use `PRIMARY KEY` right after defining the attribute in the `CREATE TABLE` statement.

There can only be one primary key per relation, other candidate keys can be specified using `UNIQUE`:

- e.g. `UNIQUE (ename)`

SQL Constraints - Referential Integrity

Referential integrity constraint - Defines a foreign key that references the primary key of another table.

- If a foreign key contains a value that is not `NULL`, that value must be present in some tuple in the relation containing the referenced primary key.

Example: `workson` contains two foreign keys:

- `workson.eno` references `emp.eno`
- `workson.pno` references `proj.pno`

Specify foreign keys using `FOREIGN KEY` syntax:

```
FOREIGN KEY (eno) REFERENCES emp(eno)
```

SQL Referential Integrity Example

The CREATE TABLE command for the workson relation:

```
CREATE TABLE workson (
    eno     CHAR(5),
    pno     CHAR(5),
    resp    VARCHAR(20),
    hours   SMALLINT,
    PRIMARY KEY (eno,pno),
    FOREIGN KEY (eno) REFERENCES emp(eno),
    FOREIGN KEY (pno) REFERENCES proj(pno)
);
```

DROP TABLE

The command **DROP TABLE** is used to delete the table definition and all data from the database:

```
DROP TABLE tableName [RESTRICT | CASCADE]
```

Example: `DROP TABLE emp;`

- Note: The database does not confirm if you really want to drop the table and delete its data. The effect of the command is immediate.
- RESTRICT will not drop object if it is used. CASCADE will drop object even if it is used.

Question: What would be the effect of the command:

```
DROP TABLE emp CASCADE;
```

Indexes

Indexes are used to speed up access to the rows of a table based on the values of certain attributes.

- An index will often significantly improve the performance of a query, however they represent an overhead as they must be updated every time the table is updated.

The general syntax for creating and dropping indexes is:

```
CREATE [UNIQUE] INDEX indexName
    ON  tableName (colName [ASC|DESC] [,...])

DROP INDEX indexName;
```

- UNIQUE means that each value in the index is unique.
- ASC/DESC specifies the sorted order of index.

Indexes Example

Creating an index on eno and pno in workson is useful as it will speed up joins with the emp and proj tables respectively.

- Index is not UNIQUE as eno (pno) can occur many times in WorksOn.

```
CREATE INDEX idxEno ON workson (eno);
CREATE INDEX idxPno ON workson (pno);
```

Most DBMSs will put an index on the primary key, but if they did not, this is what it would like for workson:

```
CREATE UNIQUE INDEX idxPK ON workson (eno,pno);
```

Adding Data using INSERT

Insert a row using the `INSERT` command:

```
INSERT INTO emp VALUES ('E9','S. Smith','1975-03-05',
                        'SA',60000,'E8','D1')
```

Fields: eno, ename, bdate, title, salary, supereno, dno

If you do not give values for all fields in the order they are in the table, you must list the fields you are providing data for:

```
INSERT INTO emp(eno, ename, salary)
         VALUES ('E9','S. Smith',60000)
```

Note: If any columns are omitted from the list, they are set to `NULL`.

INSERT Multiple Rows

INSERT statement extended by many databases to take multiple rows:

```
INSERT INTO tableName [(column list)]
VALUES (data value list) [, (values) ]*
```

Example:

```
INSERT INTO emp (eno, ename) VALUES
    ('E10', 'Fred'), ('E11', 'Jane'), ('E12', 'Joe')
```

INSERT rows from SELECT

Insert multiple rows that are the result of a SELECT statement:

```
INSERT INTO tableName [(column list)]
  SELECT ...
```

Example: Add rows to a temporary table that contains only employees with title ='EE'.

```
INSERT INTO tmpTable
    SELECT eno, ename
    FROM emp
    WHERE title = 'EE'
```

UPDATE Statement

Updating existing rows using the `UPDATE` statement. Examples:

- 1) Increase all employee salaries by 10%.

```
UPDATE emp SET salary = salary*1.10;
```

- 2) Increase salary of employee E2 to $1 million and change his name:

```
UPDATE emp SET salary = 1000000, name='Rich Guy'
WHERE eno = 'E2';
```

Notes:

- May change (`SET`) more than one value at a time. Separate by commas.
- Use `WHERE` to filter only the rows to update.

★

DELETE Statement

Rows are deleted using the DELETE statement. Examples:

- 1) Fire everyone in the company.

```
DELETE FROM emp;
```

- 2) Fire everyone making over $35,000.

```
DELETE FROM emp
WHERE salary > 35000;
```

Practice Questions

Relational database schema:

```
emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)
```

1) Insert a department with number `'D5'`, name `'Useless'`, and no manager.
2) Insert a `workson` record with `eno='E1'` and `pno='P3'`.
3) Delete all records from `emp`.
4) Delete only the records in `workson` with more than 20 hours.
5) Update all employees to give them a 20% pay cut.
6) Update the projects for `dno='D3'` to increase their budget by 10%.

SQL Queries using SELECT

A query in SQL has the form:

```
SELECT  (list of columns or expressions)
FROM    (list of tables)
WHERE  (filter conditions)
GROUP  BY  (columns)
ORDER  BY  (columns)
```

Notes:

- 1) Separate the list of columns/expressions and list of tables by **commas**.
- 2) The "*" is used to select all columns.
- 3) Only SELECT required. FROM, WHERE, GROUP BY, ORDER BY are optional.

SQL and Relational Algebra

The SELECT statement can be mapped directly to relational algebra.

SELECT $A_1, A_2, \ldots , A_n$

FROM $R_1, R_2, \ldots , R_m$

WHERE P

is equivalent to:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$$

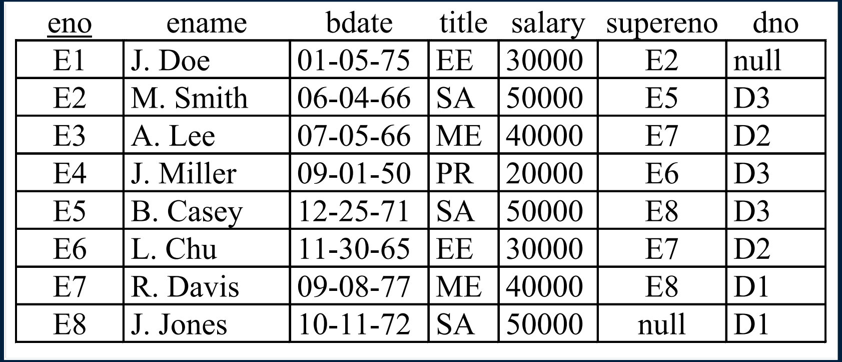
SQL: Retrieving Only Some of the Columns

The ***projection operation*** creates a new table that has some of the columns of the input table. In SQL, provide the table in the `FROM` clause and the fields in the output in the `SELECT`.

Example: Return only the `eno` field from the `emp` table:

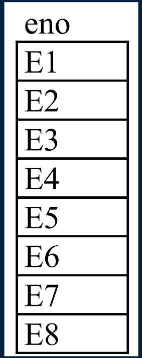
```
SELECT  eno
FROM    emp
```

emp Table



eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

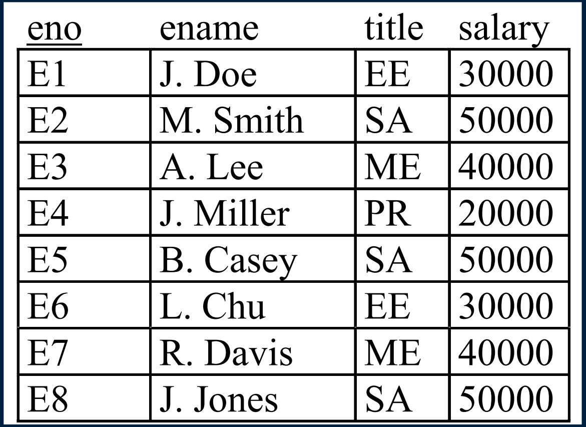
Result



eno
E1
E2
E3
E4
E5
E6
E7
E8

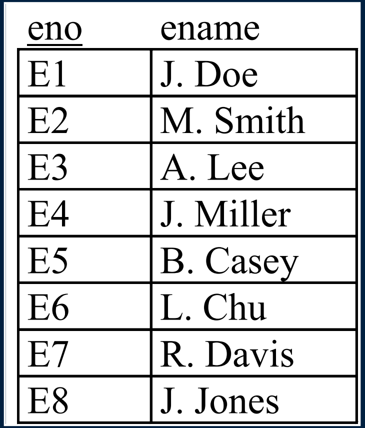
SQL Projection Examples

emp Table



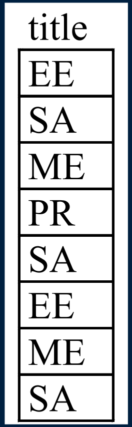
eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

```
SELECT  eno,ename
FROM    emp
```



eno	ename
E1	J. Doe
E2	M. Smith
E3	A. Lee
E4	J. Miller
E5	B. Casey
E6	L. Chu
E7	R. Davis
E8	J. Jones

```
SELECT  title
FROM    emp
```



title
EE
SA
ME
PR
SA
EE
ME
SA

Notes: 1) Duplicates are not removed during SQL projection.
2) `SELECT *` will return all columns.

Duplicates in SQL

One major difference between SQL and relational algebra is that relations in SQL are ***bags*** instead of sets.

- It is possible to have two or more identical rows in a relation.

Consider the query: Return all titles of employees.

```
SELECT title
FROM   emp
```

emp Table

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Result

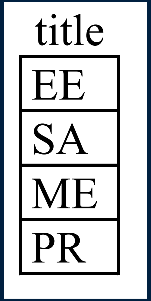
title
EE
SA
ME
PR
SA
EE
ME
SA

Duplicates in SQL - DISTINCT clause

To remove duplicates, use DISTINCT clause in the SQL statement:

```
SELECT DISTINCT title
FROM   emp
```

Result



title
EE
SA
ME
PR

DISTINCT Question

Question: Given this table and the query:

```
SELECT DISTINCT a, b
FROM     R
```

R Table

a	b	c
1	1	A
1	2	B
1	1	A
3	1	C
2	2	A
2	2	B

How many rows are returned?

A) 1

B) 3

C) 4

D) 6

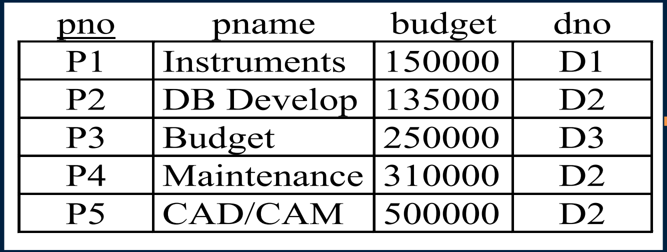
Retrieving Only Some of the Rows

The ***selection operation*** creates a new table with some of the rows of the input table. A condition specifies which rows are in the new table. The condition is similar to an `if` statement.

Example: Return the projects in department `'D2'`:

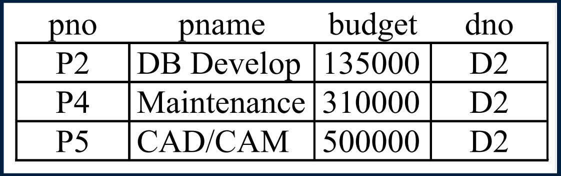
```
SELECT pno, pname, budget, dno
FROM   proj
WHERE  dno = 'D2'
```

`proj` Table



pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

Result



pno	pname	budget	dno
P2	DB Develop	135000	D2
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

Algorithm: Scan each tuple and check if matches condition in WHERE clause.

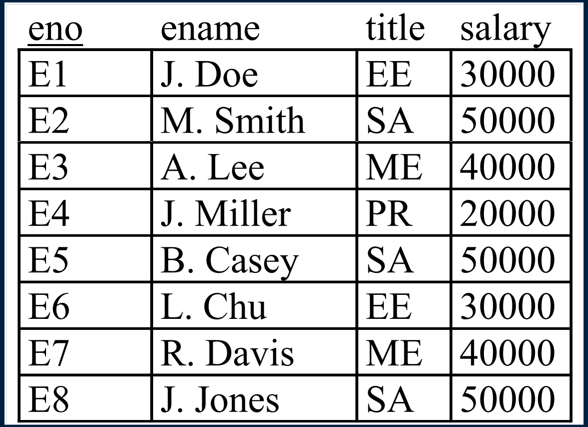
Selection Conditions

The condition in a selection statement specifies which rows are included. It has the general form of an if statement.

The condition may consist of attributes, constants, comparison operators (<, >, =, != , <=, >=), and logical operators (`AND`, `OR`, `NOT`).

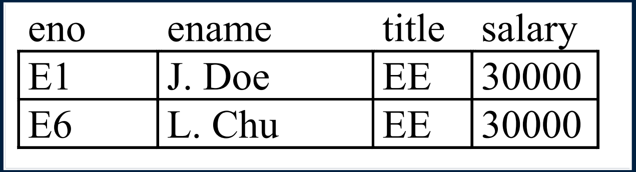
SQL Selection Examples

emp Table



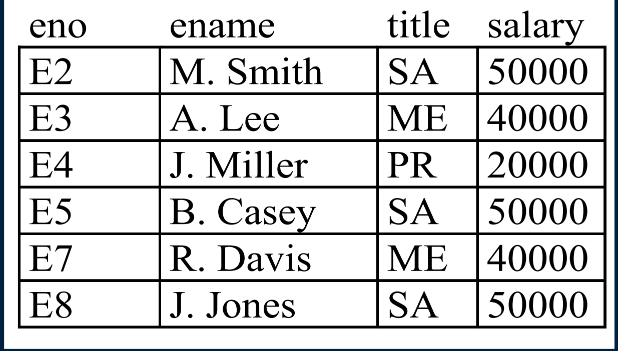
eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

```
SELECT *
FROM   emp
WHERE  title = 'EE'
```



eno	ename	title	salary
E1	J. Doe	EE	30000
E6	L. Chu	EE	30000

```
SELECT eno, ename, title, salary
FROM   emp
WHERE  salary > 35000 OR
         title = 'PR'
```



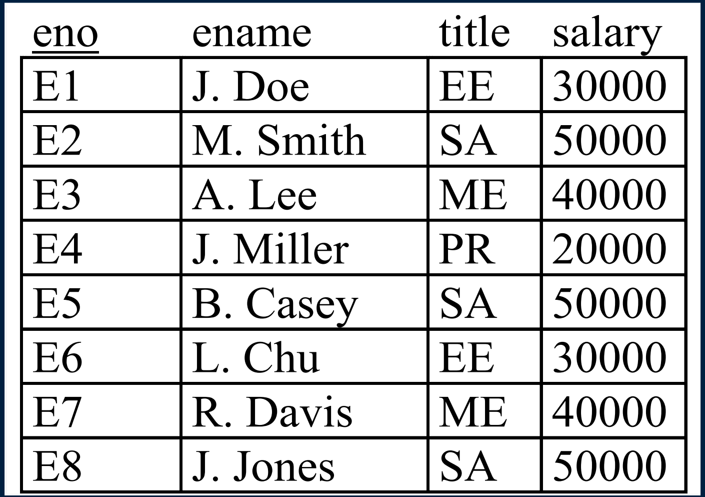
eno	ename	title	salary
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection Question #2

Question: Given this table and the query:

```
SELECT *
FROM   emp
WHERE  salary > 50000 or title='PR'
```

emp Table



eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

How many rows are returned?

A) 0

B) 1

C) 2

D) 3

Try it: SQL `SELECT` and Filtering Rows

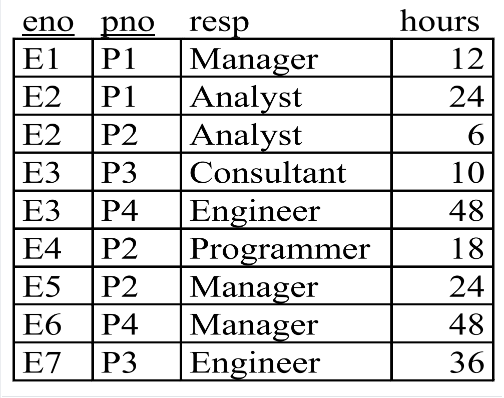
Question: Write these queries:

1) Return all projects with `budget >` $`250000`.
2) Show the `pno` and `pname` for projects in `dno = 'D1'`.
3) Show `pno` and `dno` for projects in `dno='D1'` or `dno='D2'`.
4) Return the employee numbers who make less than $`30000`.
5) Return list of `workson` responsibilities (`resp`) with no duplicates.
6) Return the employee (names) born after July 1, 1970 that have a `salary > 35000` and have a title of `'SA'` or `'PR'`.

Joins for Combining Tables

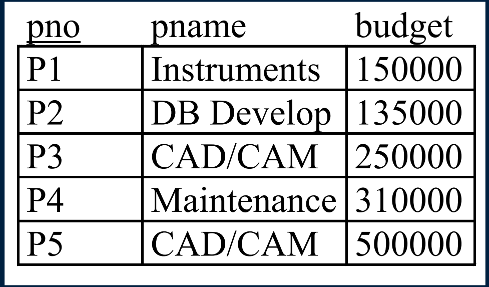
A ***join*** combines two tables by matching columns in each table.

`workson` Table



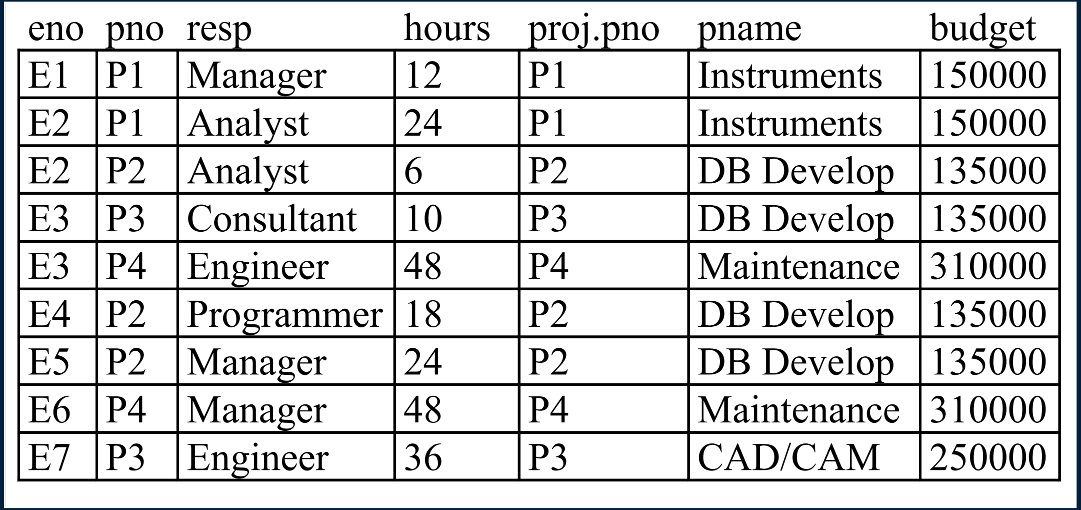
eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

`proj` Table



pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT *
FROM   workson JOIN proj
   ON  workson.pno = proj.pno
```



eno	pno	resp	hours	proj.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P3	Consultant	10	P3	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E4	P2	Programmer	18	P2	DB Develop	135000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000

Join Details and Examples

Listing multiple tables in the `FROM` clause separated by commas creates a cross product of tables. Must specify `JOIN` and `ON` or provide join condition in `WHERE` clause.

Goal: For each employee, return their name and department name.

Wrong! Cross Product

```
SELECT ename, dname
FROM   emp, dept
```

Correct! `JOIN-ON` Clause

```
SELECT ename, dname
FROM   emp JOIN dept
       ON emp.dno = dept.dno
```

Correct! Join in `WHERE`

```
SELECT ename, dname
FROM   emp, dept
WHERE  emp.dno = dept.dno
```

Correct! Order does not matter.

```
SELECT ename, dname
FROM   dept JOIN emp
       ON emp.dno = dept.dno
```

Join Query with Selection Example

You can use join, selection, and projection in the same query.

- Recall: Projection returns columns listed in `SELECT`, selection filters out rows using condition in `WHERE`, and join combines tables in `FROM` using a condition.

Example: Return the employee names who are assigned to the 'Management' department.

Projection: only name field in result

```
SELECT ename
FROM   emp JOIN dept
         ON emp.dno = dept.dno
WHERE  dname = 'Management'
```

tables in query joined together

Selection: filter rows

Result

ename
R. Davis
J. Jones

Three Table Join Query Example

Return all projects who have an employee working on them whose title is `'EE'`:

```
SELECT pname
FROM   emp JOIN workson ON emp.eno = workson.eno
           JOIN proj ON workson.pno = proj.pno
WHERE  emp.title = 'EE'
```

Or:

```
SELECT pname
FROM   emp, proj, workson
WHERE  emp.title = 'EE' and workson.eno = emp.eno
        and workson.pno = proj.pno
```

Note: Parentheses `()` can be used to specify order of joins when using `JOIN-ON`.

SQL Query Question

Question: What query would return the name and salary of employees working on project `'P3'`:

A)
```
SELECT ename, salary
 FROM emp, workson
 WHERE emp.eno = workson.eno and pno = 'P3'
```

B)
```
SELECT ename, salary
 FROM emp, workson, proj
 WHERE emp.eno = workson.eno and pno = "P3"
```

Ordering Result Data

The query result returned is not ordered on any column by default. We can order the data using the **ORDER BY** clause:

```
SELECT   ename, salary, bdate
FROM     emp
WHERE    salary > 30000
ORDER BY salary DESC, ename ASC;
```

- 'ASC' sorts the data in ascending order, and 'DESC' sorts it in descending order. The default is 'ASC'.
- The order of sorted attributes is significant. The first column specified is sorted on first, then the second column is used to break any ties, etc.

LIMIT and OFFSET

If you only want the first *N* rows, use a **LIMIT** clause:

```
SELECT   ename, salary FROM emp
ORDER BY salary DESC LIMIT 5
```

To start from a row besides the first, use**OFFSET**:

```
SELECT   eno, salary FROM emp
ORDER BY eno DESC
LIMIT 3  OFFSET 2
```

- LIMIT improves performance by reducing amount of data processed and sent by the database system.
- OFFSET 0 is first row, so OFFSET 2 would return the 3rd row.
- LIMIT/OFFSET syntax support differs between databases.

Try it: SQL `SELECT` with Joins and Ordering

Question: Write these queries:

1) Return all projects with `budget < $500000` sorted by `budget` descending.
2) List only the top 5 employees by `salary` descending. Show only their `name` and `salary`.
3) List each project `pno`, `dno`, `pname`, and `dname` ordered by `dno` ascending then `pno` ascending. Only show projects if department `name > 'D'`. Note: This query will require a join.
4) Return the list of project names for the department with name `'Consulting'`.
5) Return `workson` records (`eno`, `pno`, `resp`, `hours`) where project `budget` is > `$50000` and hours worked is `< 20`.
6) **Challenge:** Return a list of all department names, the names of the projects of that department, and the name of the manager of each department.

Calculated Fields

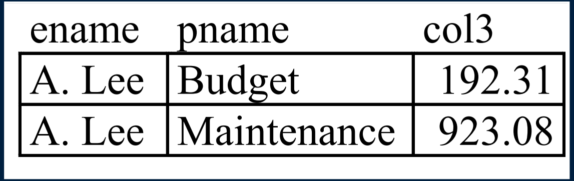
Expressions are allowed in `SELECT` clause to perform calculations.

- When an expression is used to define an attribute, the DBMS gives the attribute a unique name such as `col1`, `col2`, etc.

Example: Return how much employee 'A. Lee' will get paid for his work on each project.

```
SELECT ename, pname, salary/52/5/8*hours
FROM   emp JOIN workson ON emp.eno=workson.eno
         JOIN proj ON workson.pno=proj.pno
WHERE  ename='A. Lee'
```

Result



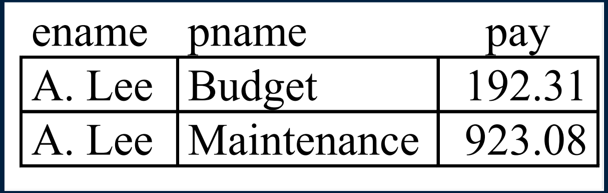
ename	pname	col3
A. Lee	Budget	192.31
A. Lee	Maintenance	923.08

Renaming and Aliasing

Often it is useful to rename an attribute in the final result (especially when using calculated fields). Renaming is accomplished using the keyword **AS**:

```
SELECT ename, pname, salary/52/5/8*hours AS pay
FROM   emp JOIN workson ON emp.eno = workson.eno
           JOIN proj ON proj.pno = workson.pno
WHERE  ename = 'A. Lee'
```

Result



ename	pname	pay
A. Lee	Budget	192.31
A. Lee	Maintenance	923.08

Note: AS keyword is optional.

Renaming and Aliasing Tables

Renaming is also used when two or more copies of the same table are in a query. Using ***aliases*** allows you to uniquely identify what table you are talking about.

Example: Return the employees and their managers where the managers make less than the employee.

```
SELECT E.ename, M.ename
FROM   emp as E JOIN emp as M ON E.supereno = M.eno
WHERE  E.salary > M.salary
```

Advanced Conditions - IN

To specify that an attribute value should be in a given set of values, the **IN** keyword is used.

- Example: Return employees who are in one of the departments {'D1', 'D2', 'D3'}.

```
SELECT ename
FROM   emp
WHERE  dno IN ('D1','D2','D3')
```

Note that this is equivalent to using OR:

```
SELECT ename
FROM   emp
WHERE  dno = 'D1' OR dno = 'D2' OR dno = 'D3'
```

We will see more uses of `IN` and `NOT IN` with nested subqueries.

Advanced Conditions - NULL

Remember NULL indicates that an attribute does not have a value. To determine if an attribute is NULL, we use the clause **IS NULL**.

- Note that you should not test NULL values using = and <>.

Example: Return all employees who are not in a department.

```
SELECT  ename
FROM    emp
WHERE   dno IS NULL
```

Example: Return all departments that have a manager.

```
SELECT  dname
FROM    dept
WHERE   mgreno IS NOT NULL
```

Aggregate Queries and Functions

Several queries cannot be answered using the simple form of the `SELECT` statement. These queries require a summary calculation to be performed. Examples:

- What is the maximum employee salary?
- What is the total number of hours worked on a project?
- How many employees are there in department 'D1'?

To answer these queries requires the use of aggregate functions. These functions operate on a single column of a table and return a single value.

Aggregate Functions

Five common aggregate functions are:

- `COUNT` - returns the # of values in a column
- `SUM` - returns the sum of the values in a column
- `AVG` - returns the average of the values in a column
- `MIN` - returns the smallest value in a column
- `MAX` - returns the largest value in a column

Notes:

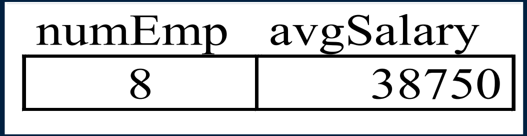
- 1) `COUNT`, `MAX`, and `MIN` apply to all types of fields, whereas `SUM` and `AVG` apply to only numeric fields.
- 2) Except for `COUNT(*)` all functions ignore nulls. `COUNT(*)` returns the number of rows in the table.
- 3) Use `DISTINCT` to eliminate duplicates.

Aggregate Function Example

Return the number of employees and their average salary.

```
SELECT COUNT(eno) AS numEmp, AVG(salary) AS avgSalary
FROM   emp
```

Result



numEmp	avgSalary
8	38750

GROUP BY Clause

Aggregate functions are most useful when combined with the GROUP BY clause. The **GROUP BY** clause groups the tuples based on the values of the attributes specified.

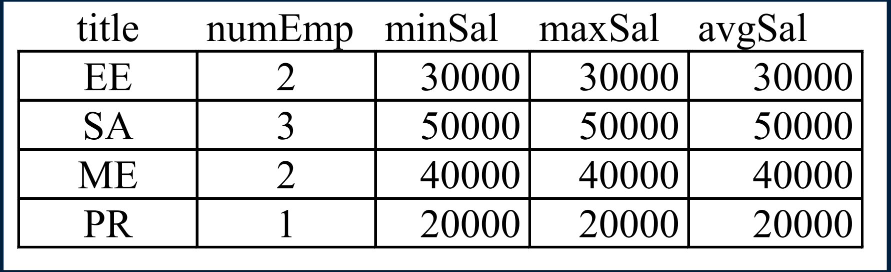
When used in combination with aggregate functions, the result is a table where each tuple consists of unique values for the group by attributes and the result of the aggregate functions applied to the tuples of that group.

GROUP BY Example

For each employee title, return the number of employees with that title, and the minimum, maximum, and average salary.

```
SELECT    title, COUNT(eno) AS numEmp,
          MIN(salary) as minSal,
          MAX(salary) as maxSal, AVG(salary) AS avgSal
FROM      emp
GROUP BY  title
```

Result



title	numEmp	minSal	maxSal	avgSal
EE	2	30000	30000	30000
SA	3	50000	50000	50000
ME	2	40000	40000	40000
PR	1	20000	20000	20000

`GROUP BY` Clause Rules

There are a few rules for using the `GROUP BY` clause:

- 1) A column name cannot appear in the `SELECT` part of the query unless it is part of an aggregate function or in the list of group by attributes.
  - Note that the reverse is allowed: a column can be in the `GROUP BY` without being in the `SELECT` part.
- 2) Any `WHERE` conditions are applied before the `GROUP BY` and aggregate functions are calculated.
- 3) You can group by multiple attributes. To be in the same group, all attribute values must be the same.

GROUP BY Question

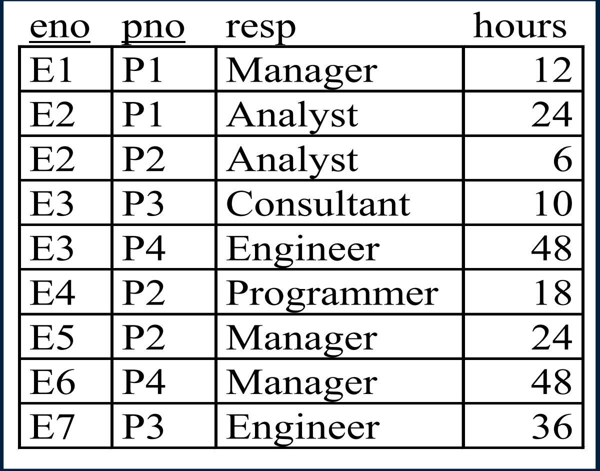
Question: Given this table and the query:

```
SELECT resp, pno, SUM(hours)
FROM   workson
WHERE  hours > 10
GROUP BY resp, pno
```

How many rows are returned?

A) 9 **B)** 7 **C)** 5 **D)** 1 **E)** 0

workson Table



eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

HAVING Clause

The **HAVING** clause is applied ***AFTER*** the GROUP BY clause and aggregate functions are calculated.

It is used to filter out entire *groups* that do not match certain criteria.

The **HAVING** clause can contain any condition that references aggregate functions and the group by attributes themselves.

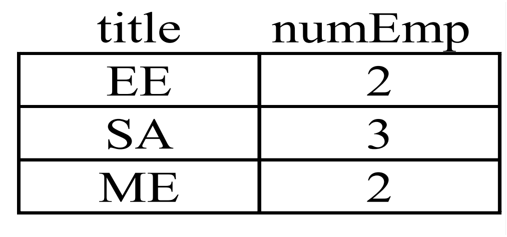
- However, any conditions on the GROUP BY attributes should be specified in the WHERE clause if possible due to performance reasons.

HAVING Example

Return the title and number of employees of that title where the number of employees of the title is at least 2.

```
SELECT    title, COUNT(eno) AS numEmp
FROM      emp
GROUP BY  title
HAVING    COUNT(eno) >= 2
```

Result



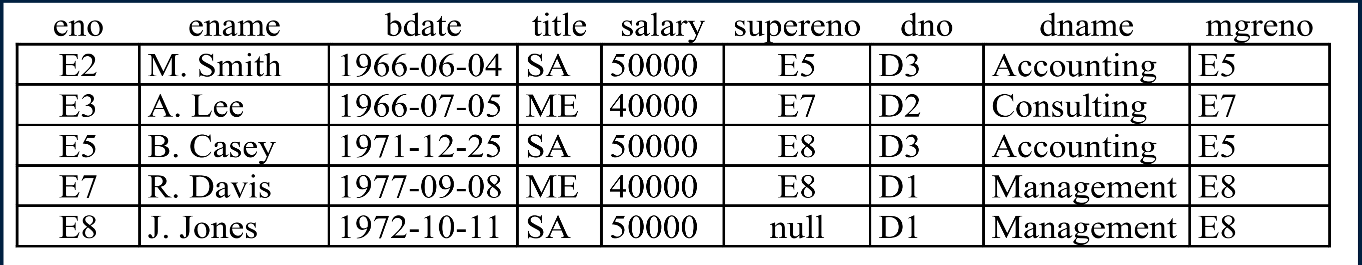
title	numEmp
EE	2
SA	3
ME	2

GROUP BY/HAVING Example

For employees born after December 1, 1965, return the average salary by department where the average is > 40,000.

```
SELECT   dname, AVG(salary) AS avgSal
FROM     emp JOIN dept ON emp.dno = dept.dno
WHERE    emp.bdate > DATE '1965-12-01'
GROUP BY dname
HAVING   AVG(salary) > 40000
```

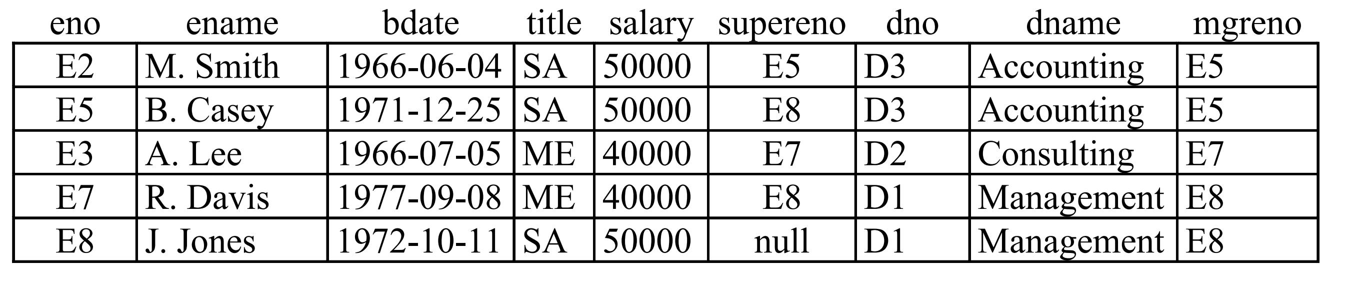
Step #1: Perform Join and Filter in WHERE clause



eno	ename	bdate	title	salary	supereno	dno	dname	mgreno
E2	M. Smith	1966-06-04	SA	50000	E5	D3	Accounting	E5
E3	A. Lee	1966-07-05	ME	40000	E7	D2	Consulting	E7
E5	B. Casey	1971-12-25	SA	50000	E8	D3	Accounting	E5
E7	R. Davis	1977-09-08	ME	40000	E8	D1	Management	E8
E8	J. Jones	1972-10-11	SA	50000	null	D1	Management	E8

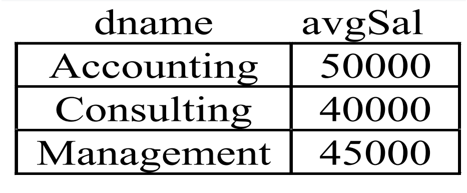
GROUP BY/HAVING Example (2)

Step #2: GROUP BY on dname



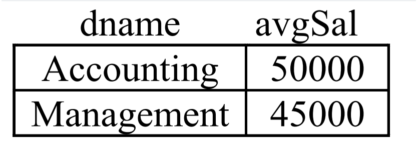
eno	ename	bdate	title	salary	supereno	dno	dname	mgreno
E2	M. Smith	1966-06-04	SA	50000	E5	D3	Accounting	E5
E5	B. Casey	1971-12-25	SA	50000	E8	D3	Accounting	E5
E3	A. Lee	1966-07-05	ME	40000	E7	D2	Consulting	E7
E7	R. Davis	1977-09-08	ME	40000	E8	D1	Management	E8
E8	J. Jones	1972-10-11	SA	50000	null	D1	Management	E8

Step #3: Calculate aggregate functions



dname	avgSal
Accounting	50000
Consulting	40000
Management	45000

Step #4: Filter groups using HAVING clause



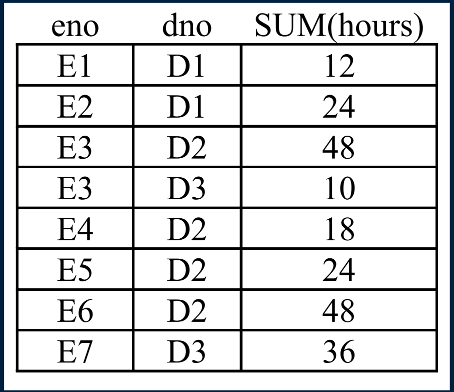
dname	avgSal
Accounting	50000
Management	45000

GROUP BY/HAVING
Multi-Attribute Example

Return the employee number, department number and hours the employee worked per department where the hours is >= 10.

```
SELECT    W.eno, D.dno, SUM(hours)
FROM      workson AS W JOIN proj AS P ON W.pno = P.pno
          JOIN dept AS D ON P.dno = D.dno
GROUP BY W.eno, D.dno
HAVING    SUM(hours) >= 10
```

Result:



eno	dno	SUM(hours)
E1	D1	12
E2	D1	24
E3	D2	48
E3	D3	10
E4	D2	18
E5	D2	24
E6	D2	48
E7	D3	36

Question:

1) How would you only return records for departments D2 and D3?

SQL Querying with `GROUP BY`

Question: Of the following queries, select one which is **invalid**.

A)
```
SELECT dname
FROM   dept
GROUP BY dno
```

B)
```
SELECT COUNT(*)
FROM   dept
```

C)
```
SELECT dno, COUNT(*)
FROM   dept
```

D)
```
SELECT dno, COUNT(*)
FROM   dept  WHERE mgreno > 'A'
GROUP BY dno, dname
```

Try it: SQL `GROUP BY` Practice Questions

Question: Write these queries:

1) Return the highest salary of any employee.

2) Return the smallest project budget.

3) Return the department number and average budget for its projects.

4) For each project, return its name and the total number of hours employees have worked on it.

5) For each employee, return the total number of hours they have worked. Only show employees with more than 30 hours.

Subqueries

SQL allows a single query to have multiple subqueries nested inside of it. This allows for more complex queries to be written.

When queries are nested, the outer statement determines the contents of the final result, while the inner `SELECT` statements are used by the outer statement (often to lookup values for `WHERE` clauses).

```
SELECT   ename, salary, bdate
FROM     emp
WHERE    salary > (SELECT AVG(salary) FROM emp)
```

A subquery can be in the `SELECT`, `FROM`, `WHERE` or `HAVING` clause.

Types of Subqueries

There are three types of subqueries:

- 1) ***scalar subqueries*** - return a single value. Often value is then used in a comparison.
    - If query is written so that it expects a subquery to return a single value, and if it returns multiple values or no values, a run-time error occurs.
- 2) ***row subquery*** - returns a single row which may have multiple columns.
- 3) ***table subquery*** - returns one or more columns and multiple rows.

Scalar Subquery Examples

Return the employees that are in the 'Accounting' department:

```
SELECT ename
FROM   emp
WHERE  dno = (SELECT dno FROM dept
               WHERE dname = 'Accounting')
```

Return all employees who work more hours than average on a single project:

```
SELECT ename
FROM   emp JOIN workson ON workson.eno = emp.eno
WHERE  workson.hours > (SELECT AVG(hours) FROM workson)
```

Table Subqueries

A table subquery returns a relation. There are several operators that can be used:

- `EXISTS` *R* - true if *R* is not empty
- *s* `IN` *R* - true if *s* is equal to one of the values of *R*
- *s* > `ALL` *R* - true if *s* is greater than **every** value in *R*
- *s* > `ANY` *R* - true if *s* is greater than **any** value in *R*

Notes:

- 1) Any of the comparison operators (<, <=, =, etc.) can be used.
- 2) The keyword `NOT` can proceed any of the operators.
  - Example: *s* `NOT IN` *R*

Table Subquery Examples

Return all departments who have a project with a budget greater than $300,000:

```
SELECT dname FROM dept WHERE dno IN
 (SELECT dno FROM proj WHERE budget > 300000)
```

Return all projects that 'J. Doe' works on:

```
SELECT pname FROM proj WHERE pno IN
      (SELECT pno FROM workson WHERE eno =
         (SELECT eno FROM emp WHERE ename = 'J. Doe'))
```

EXISTS Example

The EXISTS function is used to check whether the result of a nested query is empty or not.

- EXISTS returns true if the nested query has 1 or more tuples.

Example: Return all employees who have the same name as someone else in the company.

```
SELECT ename
FROM   emp as E
WHERE  EXISTS (SELECT * FROM emp as E2
               WHERE E.ename = E2.ename AND
                     E.eno <> E2.eno)
```

ANY and ALL Example

ANY means that any value returned by the subquery can satisfy the condition.

ALL means that all values returned by the subquery must satisfy the condition.

Example: Return the employees who make more than all the employees with title 'ME' make.

```
SELECT ename
FROM   emp as E
WHERE  salary > ALL (SELECT salary FROM emp
                     WHERE title = 'ME')
```

Subquery Syntax Rules

1) The `ORDER BY` clause may not be used in a subquery.

2) The number of attributes in the `SELECT` clause in the subquery must match the number of attributes compared to with the comparison operator.

3) Column names in a subquery refer to the table name in the `FROM` clause of the subquery by default. You must use aliasing if you want to access a table that is present in both the inner and outer queries.

Correlated Subqueries

Most queries involving subqueries can be rewritten so that a subquery is not needed.

- This is normally beneficial because query optimizers may not do a good job at optimizing queries containing subqueries.

A nested query is ***correlated*** with the outside query if it must be re-computed for every tuple produced by the outside query. Otherwise, it is ***uncorrelated***, and the nested query can be converted to a non-nested query using joins.

A nested query is correlated with the outer query if it contains a reference to an attribute in the outer query.

Correlated Subquery Example

Return all employees who have the same name as another employee:

```
SELECT ename
FROM   emp as E
WHERE  EXISTS (SELECT eno FROM emp as E2
               WHERE E.ename = E2.ename AND
                     E.eno <> E2.eno)
```

A more efficient solution with joins:

```
SELECT E.ename
FROM   emp as E JOIN emp as E2 ON
       E.ename = E2.ename AND E.eno <> E2.eno
```

Types of Joins

A ***equijoin*** only contains the equality operator (=).

- e.g. `workson JOIN Proj ON workson.pno=proj.pno`

A ***natural join*** is equijoin of two tables with commonly named fields.

- Removes the "extra copies" of the join attributes.
- The attributes must have the same name in both relations.
- e.g. `workson NATURAL JOIN proj`

Left outer join – contains all tuples of first table even if no match

Right outer join – contains all tuples of second table even if no match

Full outer join – contains all tuples of either table even if no match. For a tuple that does not have a match, missing fields are `NULL`.

Specifying Outer Joins in SQL

Types: NATURAL JOIN, FULL OUTER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, INNER JOIN, JOIN

- The keyword "outer" can be omitted for outer joins. Same with "inner".

Example: Return all departments (even those without projects) and their projects.

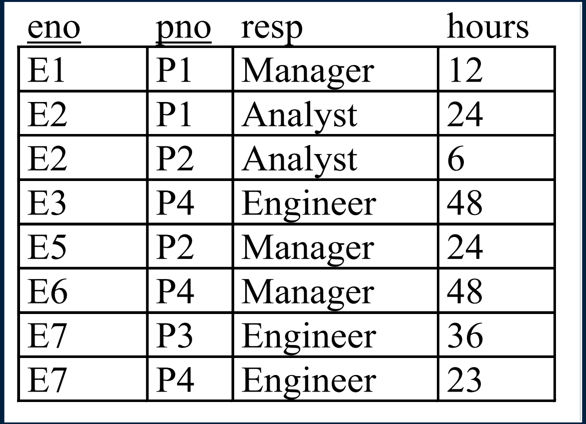
```
SELECT dname, pname
FROM   dept LEFT OUTER JOIN proj ON dept.dno = proj.dno
```

```
SELECT dname, pname
FROM   dept LEFT OUTER JOIN proj USING (dno)
```

```
SELECT dname, pname
FROM   dept NATURAL LEFT JOIN proj
```

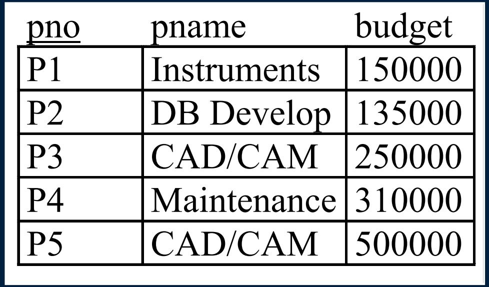
Equijoin Example

`workson` Table



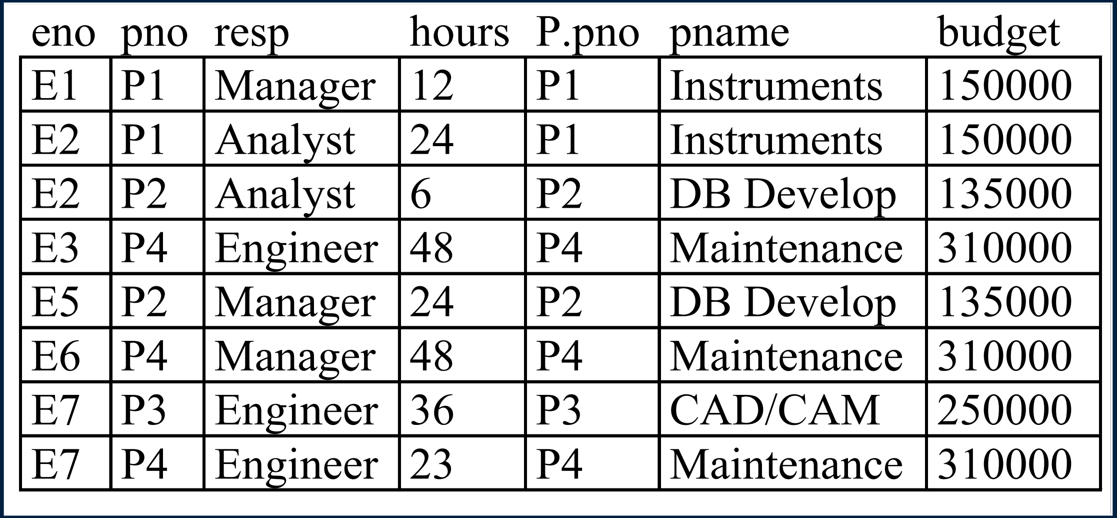
eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

`proj` table



pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT *
FROM    workson JOIN proj
    ON  workson.pno = proj.pno
```

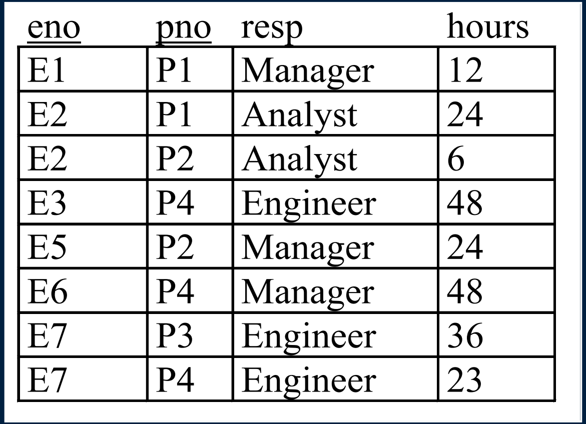


eno	pno	resp	hours	P.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P4	Maintenance	310000

What is the meaning of this join?

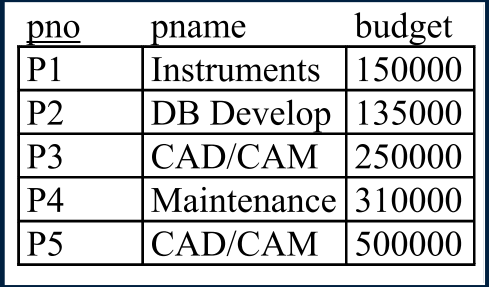
Natural Join Example

workson Table



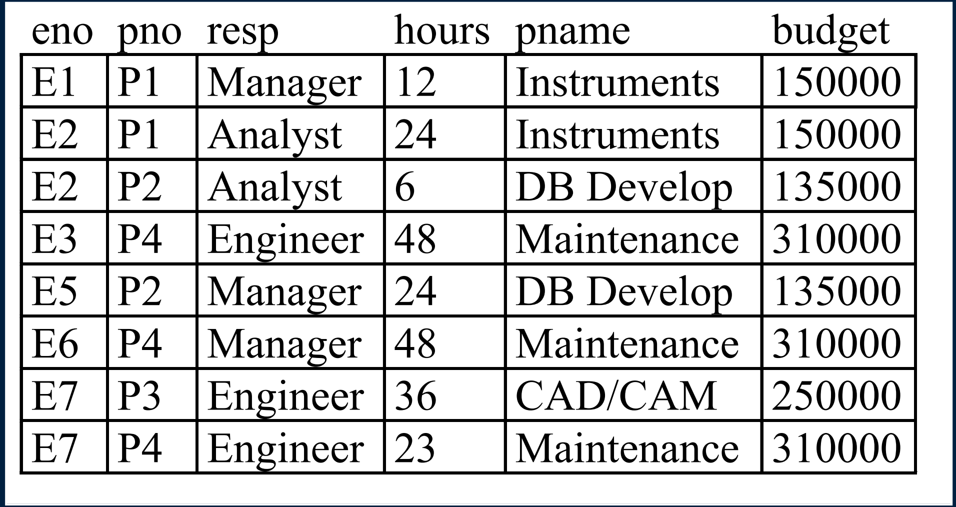
eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

proj table



pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT  *
FROM    workson NATURAL JOIN proj
```

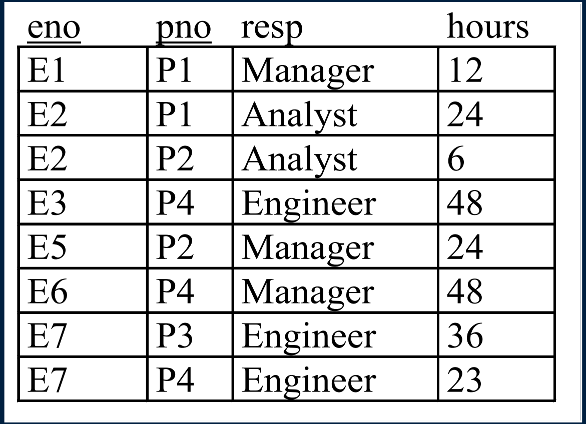


eno	pno	resp	hours	pname	budget
E1	P1	Manager	12	Instruments	150000
E2	P1	Analyst	24	Instruments	150000
E2	P2	Analyst	6	DB Develop	135000
E3	P4	Engineer	48	Maintenance	310000
E5	P2	Manager	24	DB Develop	135000
E6	P4	Manager	48	Maintenance	310000
E7	P3	Engineer	36	CAD/CAM	250000
E7	P4	Engineer	23	Maintenance	310000

Natural join is performed by comparing *pno* in both tables.

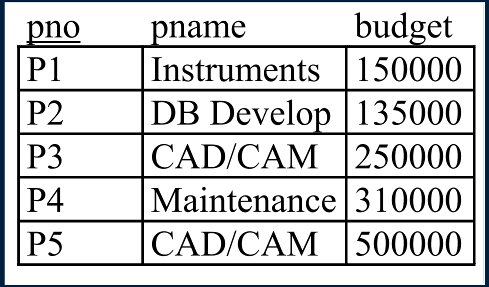
Right Outer Join Example

workson Table



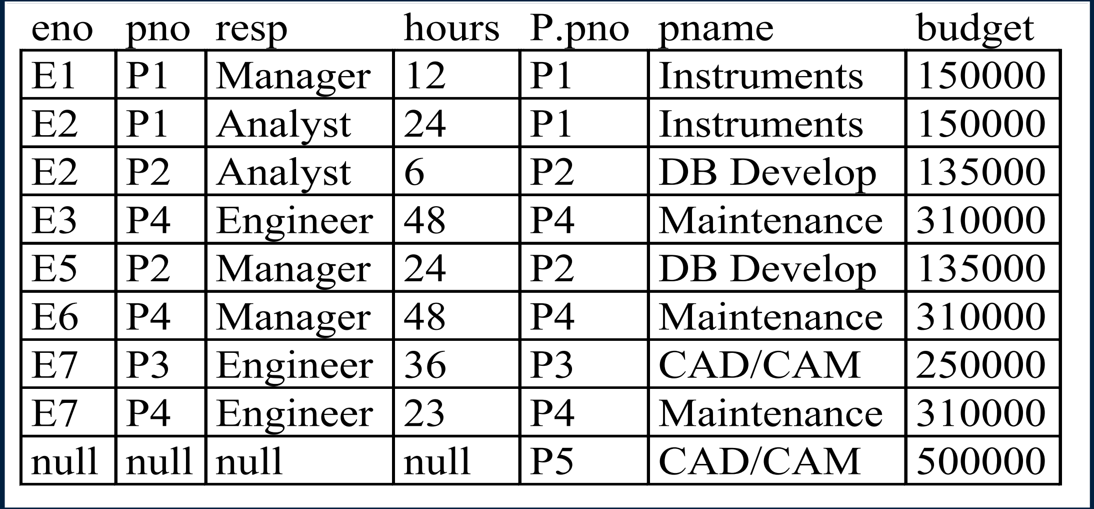
eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

proj table



pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT *
FROM workson RIGHT OUTER JOIN proj P
   USING (pno)
```



eno	pno	resp	hours	P.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P4	Maintenance	310000
null	null	null	null	P5	CAD/CAM	500000

Outer Join Question

Question: Given this table and the query:

```
SELECT *
FROM workson LEFT OUTER JOIN proj P
   ON workson.pno = proj.pno
```

How many rows are returned?

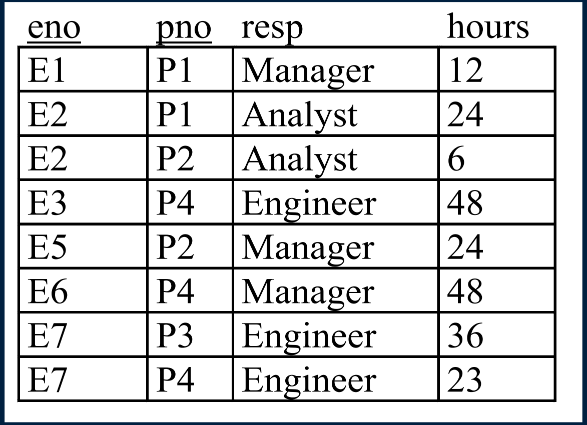
A) 10

B) 9

C) 8

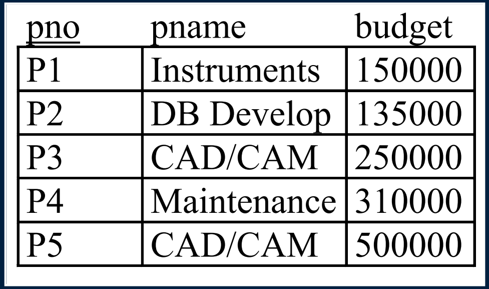
D) 7

workson



eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

proj



pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

Subqueries in FROM Clause

Subqueries are used in the FROM clause to produce temporary table results for use in the current query.

Example: Return the departments that have an employee that makes more than $40,000.

```
SELECT dname
FROM   Dept D, (SELECT ename, dno FROM Emp
               WHERE salary > 40000) E
WHERE D.dno = E.dno
```

- Note: The alias for the derived table is required.

SQL Querying with Subqueries

Question: What query below is equivalent to:

```
SELECT ename
FROM   emp as E
WHERE  salary > ALL (SELECT salary
                     FROM emp WHERE title = 'EE')
```

A)

```
SELECT ename
FROM   emp as E
WHERE  salary > (SELECT MAX(salary) FROM emp
                 WHERE title = 'EE')
```

B)

```
SELECT ename
FROM   emp as E
WHERE  salary > (SELECT SUM(salary) FROM emp
                 WHERE title = 'EE')
```

SQL Functions

Databases have many built-in functions that can be used when writing queries. Syntax and support varies between systems.

- Date: DATEDIFF, YEAR, GETDATE
- String: CONCAT, UPPER, LEFT, SUBSTRING
- Logical: CASE, IIF, ISNULL
- Aggregate: SUM, COUNT, AVG
- Note: Case-insensitive function names.

Example:

```
SELECT eno, UPPER(ename),
   CASE title WHEN 'EE' THEN 'Engineer'
   WHEN 'SA' THEN 'Admin' ELSE 'Other' END as role,
   year(bdate) as birthYear
FROM emp
WHERE salary * 2 > 60000
```

Try-it: Subquery Practice Questions

Question: Write these queries:

1) List all departments that have at least one project.

2) List the employees who are not working on any project.

3) List the employees with title 'EE' that make more than all employees with title 'PR'.

4) Find all employees who work on some project that 'J. Doe' works on.

SQL Queries using `SELECT`

A query in SQL has the form:

`SELECT` **(list of columns or expressions)**

`FROM` **(list of tables)**

`WHERE` **(filter *conditions*)**

`GROUP BY` **(columns)**

`HAVING` **(group filter *conditions*)**

`ORDER BY` **(columns)**

`LIMIT` **(count)** **`OFFSET`** **(start)**

Conclusion

SQL is the standard query language for databases. SQL contains a data definition language to `CREATE`, `ALTER`, and `DROP` database objects such as tables, indexes, schemas, and views.

Constraints preserve the integrity of the database:

- **entity integrity constraint** (primary keys) and **referential integrity constraint** (foreign keys).

`INSERT`, `DELETE`, and `UPDATE` commands modify the data stored within the database.

`SELECT` statement queries data and combines the operations of selection, projection, and join.

- **`SELECT`** clause to provide column list and calculate expressions/functions
- **`DISTINCT`** clause to eliminate duplicates
- **`FROM`** clause to list tables
- **`JOIN ON`** syntax to join tables on a join condition and perform outer/natural joins
- **`IS NULL`** for checking if column value is null
- **`ORDER BY`** clause for sorting output
- **`LIMIT/OFFSET`** for only retrieving a part of the result set
- **`GROUP BY`** for grouping data and calculating aggregate functions
- **`HAVING`** for filtering groups

Objectives

SQL DDL:

- Recognize valid and invalid identifiers.
- Explain required (not null) data, domain constraints, entity integrity, referential integrity.
- Write a `CREATE TABLE` statement given a high-level description.
- Remove a table using `DROP TABLE`.
- Create an index on fields of a table.

Write `INSERT`, `DELETE`, and `UPDATE` commands.

Translate English questions into SQL queries that may require:

- `SELECT-FROM-WHERE` syntax for selection, projection, and join
- renaming and aliasing including queries with multiple copies of the same relation
- `ORDER BY`
- `LIMIT/OFFSET`
- `DISTINCT` to eliminate duplicates
- `IS NULL` or `IS NOT NULL`
- `GROUP BY` and aggregate functions and calculated fields including SQL functions
- Subqueries and operators such as `IN`, `NOT IN`, `ANY`, `ALL` , `EXISTS`, `NOT EXISTS`
- `OUTER` and `NATURAL` joins

THE UNIVERSITY OF BRITISH COLUMBIA