History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
- Not all examples here may work on your particular system.

SQL

- The SQL language has several aspects to it:

- **The Data Definition Language (DDL):**
  - This subset of SQL supports the creation, deletion, and modification of definitions for tables and views.

  - *Integrity constraints* can be defined on tables, either when the table is created or later.

  - The DDL also provides commands for specifying *access rights* or *privileges* to tables and views.

- **The Data Manipulation Language (DML):**
  - This subset of SQL allows users to pose queries and to insert, delete, and modify rows.

- **Embedded and dynamic SQL:**
  - Embedded SQL features allow SQL code to be called from a host language such as C or COBOL.

- **Triggers:**
  - The new SQL:1999 standard includes support for *triggers*, which are actions executed by the DBMS whenever changes to the database meet conditions specified in the trigger.

- **Security:**
  - SQL provides mechanisms to control users' access to data objects such as tables and views.

- **Transaction management:**
  - Various commands allow a user to explicitly control aspects of how a transaction is to be executed.

- **Client-server execution and remote database access:**
  - These commands control how a *client* application program can connect to an SQL database *server*, or access data from a database over a network.

Data Definit ion Language (DDL)

- CREATE
- ALTER
- DROP
- TRUNCATE

Create command

- **Creating a Database**
  - create database database-name;
  - create database stud-db;

- **Creating a Table**

```
create table table_name
{
 column-name1 datatype1,
 column-name2 datatype2,
 column-name3 datatype3,
 column-name4 datatype4
and constraints
};
```

- **Data Types**
  - number(p, s)
  - char (size)
  - Varchar2(size)
  - Date

- Constraints are
  - Primary key
    - Primarykey(column_name)
  - Foreign key
    - Foreign key (column_name)references table_name(column_name)
  - Unique
    - Unique(Column_name)
  - Check
    - Check(search_condition)

alter command

- alter command is used for alteration of table structures

- **To Add Column to existing Table**
  - alter table table-name add(column-name datatype);
  - alter table Student add(address char);

- **To Modify an existing Column**
  - alter table table-name modify(column-name datatype);
  - alter table Student modify(address varchar(30));

- **To Rename a column**
  - alter table table-name rename old-column-name to column-name;
  - alter table Student rename address to Location;

- **To Drop a Column**
  - alter table table-name drop(column-name);
  - alter table Student drop(address);

- **The DROP Command**
- It will destroy the table and all data which will be recorded in it
  - DROP TABLE <table_name>
  - DROP TABLE Student;
- **The TRUNCATE Command**
- The above query will delete all the records of a table
  - TRUNCATE TABLE <Table_name>
  - TRUNCATE TABLE Student;
- **rename query**
- *rename* command is used to rename a table
  - **rename table old-table-name to new-table-name**
  - **rename table Student to Student-record;**

TCL command

- Transaction Control Language(TCL) commands are used to manage transactions in database
- **Commit command**
  - Commit command is used to permanently save any transaction into database.
  - commit;
- **Rollback command**
  - This command restores the database to last committed state.
  - It is also use with savepoint command to jump to a savepoint in a transaction.
  - rollback to savepoint-name;
- **Savepoint command**
  - **savepoint** command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

DML commands

- INSERT INTO
- UPDATE
- DELETE FROM
- SELECT

- **INSERT command**
  - INSERT into table-name values(data1,data2,..)
  - INSERT into Student values(101,'Adam',15);

  - INSERT into table_name(column1,column2,….) values(data1,data2,……);
  - INSERT into Student(id,name) values(102,'Alex’);
- **UPDATE command**
  - UPDATE table-name set column-name = value where condition;
  - UPDATE Student set s_name='Abhi',age=17 where s_id=103;
- **Delete command**
  - DELETE from Student where condition;
  - DELETE from Student where s_id=103;

Examples

- Sailors(sid: integer, sname: string, rating: integer, age: real)
- Boats(bid: integer, bname: string, color: string)
- Reserves(sid: integer, bid: integer, day: date)

Basic SQL Query

SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*

- ***relation-list*** A list of relation names (possibly with a *range-variable* after each name).
- ***target-list*** A list of attributes of relations in *relation-list qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of)combined using AND, OR and NOT

- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Boats

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.

- *Q) Find the names and ages of all sailors.*
- SELECT DISTINCT S.sname, S.age FROM Sailors S

With DISTINCT

sname	*age*
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Art	25.5
Bob	63.5

Without DISTINCT

sname	*age*
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Horatio	35.0
Art	25.5
Bob	63.5

- *Find all sailors with a rating above 7.*

```
SELECT S.sid, S.sname, S.rating, S.age
FROM Sailors AS S
WHERE S.rating > 7
```

- *Find the names of sailors who have reserved boat number 103*

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid=103
```

OR

```
SELECT sname
FROM Sailors, Reserves
WHERE Sailors.sid=Reserves.sid
AND bid=103
```

Sailors S

sid	*sname*	*rating*	*age*
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Reserves R

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

S× R

sid	*sname*	*rating*	*age*	*sid*	*bid*	*day*
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Result

sname
rusty

- Find the sids of sailors who have reserved a red boat

```
Select  R.sid
FROM Boats B, Reserves R
WHERE B.bid = R.bid AND B.color = `red'
```

- *Find the names of sailors who have reserved a red boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
```

- *Find the colors of boats reserved by Lubber*

```
SELECT B.color
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND S.sname = `Lubber'
```

- *Find the names of sailors who have reserved at least one boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid
```

Expressions and Strings in the SELECT Command

- Each item in a **select-list** can be of the form
  - *expression* AS *column name*,
  - where *expression* is any arithmetic or string expression over column names and constants

- *Compute the increments ratings of persons who have sailed two different boats on the same day.*

```
SELECT S.sname, S.rating+1 AS rating
FROM Sailors S, Reserves R1,Reserves R2
Where S.sid=R1.sid AND S.sid=R2.sid AND R1.day=R2.day AND R1.bid<>R2.bid;
```

String Operations

- SQL includes a string-matching operator for comparisons on character strings.
- The operator “like” uses patterns that are described using two special characters:
- percent (%). The % character matches any substring(%’ stands for 0 or more characters)
- underscore (_). The _ character matches any character.(`_’ stands for any one character)

Ordering the Display of Tuples

SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*
Order by column_name sort_order;

- *sort_order may be either* ***desc*** *or* ***asc***

- *Find the ages of sailors whose name begins and ends with B and has at least three characters.*

- SELECT S.age
- FROM Sailors S
- WHERE S.sname LIKE `B %B'

- *Find the names of sailors who have reserved a red or a green boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND (B.color = `red' OR B.color = `green')
```

- *Find the names of sailors who have reserved both a red and a green boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R1, Boats B1, Reserves R2, Boats B2
WHERE S.sid = R1.sid AND R1.bid = B1.bid
AND S.sid = R2.sid AND R2.bid = B2.bid
AND B1.color=`red' AND B2.color = `green'
```

UNION, INTERSECT, AND EXCEPT

- SQL provides three set-manipulation constructs that extend the basic query form presented earlier

UNION
INTERSECT
EXCEPT

- *Find the names of sailors who have reserved a red or a green boat*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
UNION
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = `green‘
```

- *Find the names of sailors who have reserved both a red and a green boat*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
INTERSECT
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = `green'
```

- *Find the sids of all sailors who have reserved red boats but not green boats.*

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid = B.bid AND B.color = `red'
EXCEPT
SELECT R2.sid
FROM Boats B2, Reserves R2
WHERE R2.bid = B2.bid AND B2.color = `green'
```

- *Find all sids of sailors who have a rating of 10 or have reserved boat 104*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating = 10
UNION
SELECT R.sid
FROM Reserves R
WHERE R.bid = 104
```

- SQL also provides other set operations:
  - IN -to check if an element is in a given set
  - op ANY, op ALL -to compare a value with the elements in a given set, using comparison operator op
  - EXISTS - to check if a set is empty

NESTED QUERIES

- A **nested query is a query** that has another query embedded within it
- the embedded query is called a **sub query.**
- A sub query typically appears within the WHERE clause of a query
- Sub queries can sometimes appear in the FROM clause or the HAVING clause

- *Find the names of sailors who have reserved boat 103.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECT R.sid
                 FROM Reserves R
                 WHERE R.bid = 103 )
```

- *Find the names of sailors who have reserved a red boat.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECT R.sid
                   FROM Reserves R
                   WHERE R.bid IN ( SELECT B.bid
                                    FROM Boats B
                                    WHERE B.color = `red' ))
```

- *Find the names of sailors who have not reserved a red boat.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN ( SELECT R.sid
                     FROM Reserves R
                     WHERE R.bid IN ( SELECT B.bid
                                      FROM Boats B
                                      WHERE B.color = `red' ))
```

Correlated Nested Queries

- In the nested queries that we have seen thus far, the inner sub query has been completely independent of the outer query

- In general the inner sub query could depend on the row that is currently being examined in the outer query

- *Find the names of sailors who have reserved boat number 103*

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS ( SELECT *
                FROM Reserves R
                WHERE R.bid = 103
                AND R.sid = S.sid )
```

- The EXISTS operator is another set comparison operator, such as IN
  - It allows us to test whether a set is nonempty
  - for each Sailor row *S, we test whether the set* of Reserves rows *R such that R.bid = 103 AND S.sid = R.sid is nonempty.*
  - If so, sailor *S has reserved boat 103, and we retrieve the name*

- The sub query clearly depends on the current row *S and must be re-evaluated for each row in Sailors.*

- *The occurrence* of *S in the sub query (in the form of the literal S.sid) is called a correlation, and such* queries are called *correlated queries*

- using NOT EXISTS instead of EXISTS, we can compute the names of sailors who have not reserved a red boat

Set-Comparison Operators

- SQL also supports op ANY and op ALL,
  - where op is one of the arithmetic comparison operators
  - *<;<=;=; <>;>=;>*
  - *SOME is also available,* but it is just a synonym for ANY

- *Find sailors whose rating is better than some sailor called Horatio*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
                       FROM Sailors S2
                       WHERE S2.sname = `Horatio' );
```

- *Find sailors whose rating is better than every sailor called Horatio.*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
                       FROM Sailors S2
                       WHERE S2.sname = `Horatio' );
```

- *Find the sailors with the highest rating.*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating >= ALL ( SELECT S2.rating
                        FROM Sailors S2 )
```

- Find the names of sailors who have reserved both a red boat and a green boat

```
Select s.sname
from sailors s
Where s.sid IN((select R.sid
                from Boats B, Reserves R
                where R.bid=B.bid AND B.color='red')
               INTERSECT
               (select R.sid
                from Boats B2, Reserves R2
                where R2.bid=B2.bid AND B2.color='green'));
```

- Find the Names of sailors who have reserved all boats

```
Select s.sname
from sailors s
Where NOT EXISTS(select B.bid
                 from Boats B
                 where NOT EXISTS( select R.bid
                                   from Reserves R
                                   where R.bid=B.bid
                                   AND   R.sid=S.sid));
```

AGGREGATE OPERATORS

- SQL supports five aggregate operations, which can be applied on any column, say A, of a relation

1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
4. MAX (A): The maximum value in the A column.
5. MIN (A): The minimum value in the A column.

- *Find the average age of all sailors.*
    - SELECT AVG (S.age)
    - FROM Sailors S
- *Find the average age of sailors with a rating of 10.*
    - SELECT AVG (S.age)
    - FROM Sailors S
    - WHERE S.rating = 10

- *Find the name and age of the oldest sailor.*
  - SELECT S.sname, MAX (S.age)
  - FROM Sailors S
- if the SELECT clause uses an aggregate operation, then it must use *only aggregate operations unless* the query contains a GROUP BY clause

- SELECT S.sname, S.age
- FROM Sailors S
- WHERE (SELECT MAX (S2.age)
- FROM Sailors S2) = S.age

- *Count the number of sailors.*
  - SELECT COUNT (*)
  - FROM Sailors S;

- *Count the number of different sailor names.*
  - SELECT COUNT (DISTINCT S.sname)
  - FROM Sailors S

- *Find the names of sailors who are older than the oldest sailor with a rating of 10.*
  - SELECT S.sname
  - FROM Sailors S
  - WHERE S.age > *(SELECT MAX (S2.age)*
  - FROM Sailors S2
  - WHERE S2.rating = 10)

  OR
  - SELECT S.sname
  - FROM Sailors S
  - WHERE S.age > *ALL (SELECT S2.age*
  - FROM Sailors S2
  - WHERE S2.rating = 10)

The GROUP BY and HAVING Clauses

- Often we want to apply aggregate operations to each of a number of groups of rows in a relation, where the number of groups depends on the relation instance

  - SELECT [DISTINCT] select-list
  - FROM from-list
  - WHERE qualification
  - GROUP BY grouping-list
  - HAVING group-qualification

- The select-list in the SELECT clause consists of
  - (1) a list of column names and
  - (2) a list of terms having the form aggop (*column-name) AS new-name*.

- Every column that appears in (1) must also appear in grouping-list.
  - The reason is that each row in the result of the query corresponds to one *group*, *which is a* collection of rows that agree on the values of columns in grouping-list.

  - If a column appears in list (1), but not in grouping-list, it is not clear what value should be assigned to it in an answer row.

- The expressions appearing in the group-qualification in the HAVING clause must have a *single value per group.*
    - a column appearing in the group-qualification must appear as the argument to an aggregation operator, or it must also appear in grouping-list.

- If the GROUP BY clause is omitted, the entire table is regarded as a single group.

- *Find the age of the youngest sailor who is eligible to vote*
  - SELECT S.rating, MIN (S.age) AS minage
  - FROM Sailors S
  - WHERE S.age *>= 18*
  - GROUP BY S.rating
  - HAVING COUNT (*) *> 1*

- *For each red boat, find the number of reservations for this boat.*
  - SELECT B.bid, COUNT (*) AS sailorcount
  - FROM Boats B, Reserves R
  - WHERE R.bid = B.bid
  - GROUP BY B.bid
  - HAVING B.color = `red'

- *Find the average age of sailors for each rating level that has at least two sailors.*
  - SELECT S.rating, AVG (S.age) AS avgage
  - FROM Sailors S
  - GROUP BY S.rating
  - HAVING COUNT (*) *> 1*

- *Find the average age of sailors who are of voting age (i.e., at least 18 years old)for each rating level that has at least two sailors.*

```
SELECT S.rating, AVG ( S.age ) AS avgage
FROM Sailors S
WHERE S. age >= 18
GROUP BY S.rating
HAVING 1 < ( SELECT COUNT (*)
FROM Sailors S2
WHERE S.rating = S2.rating )
```

- *Find those ratings for which the average age of sailors is the minimum over all ratings.*

```
SELECT Temp.rating, Temp.avgage
FROM ( SELECT S.rating, AVG (S.age) AS avgage,
FROM Sailors S
GROUP BY S.rating) AS Temp
WHERE Temp.avgage = ( SELECT MIN (Temp.avgage) FROM Temp )
```

NULL VALUES

- SQL provides a special column value called *null* .
- *We use null when the column value is either unknown or inapplicable.*
- The presence of *null values complicates many issues, and we consider the impact of null values*

Comparisons Using Null Values

- SQL also provides a special comparison operator IS NULL to test whether a column value is *null;*
- IS NOT NULL

Logical Connectives AND, OR, and NOT

- we must dene the logical operators AND, OR, and NOT using a *three-valued logic in which expressions* evaluate to true, false, or unknown.

- OR of two arguments evaluates to true if either argument evaluates to true, and to unknown if one argument evaluates to false and the other evaluates to unknown

- AND of two arguments evaluates to false if either argument evaluates to false, and to unknown if one argument evaluates to unknown and the other evaluates to true or unknown.

Outer Joins

- Some interesting variants of the join operation that rely on *null values*, *called* ***outer* joins**
- Consider the join of two tables, say Sailors $\bowtie_c$ *Reserves*

- Tuples of Sailors that do not match some row in Reserves according to the join condition *c do not appear in the result.*

- Sailor rows without a matching Reserves row appear exactly once in the result, with the result columns inherited from Reserves assigned *null values.*

- a **left outer join, Sailor** rows without a matching Reserves row appear in the result, but not vice versa

- In a **right outer join, Reserves rows without a matching Sailors row appear in the result,** but not vice versa

- In a **full outer join, both Sailors and Reserves rows without a** match appear in the result.

- SELECT Sailors.sid, Reserves.bid
- FROM Sailors NATURAL LEFT OUTER JOIN Reserves R

- The NATURAL keyword species that the join condition is equality on all common attributes

sid	*sname*	*rating*	*age*
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Sailors

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

Reserves

sid	*bid*
22	101
31	*null*
58	103

Disallowing Null Values

- We can disallow *null values by specifying NOT NULL as part of the field definition,*
- *for* example, *sname CHAR(20) NOT NULL*
- the fields in a primary key are not allowed to take on *null values*

INTRODUCTION TO VIEWS

A **view is a table whose rows are not explicitly stored in the database but are computed** as needed from a **view definition.**

CREATE VIEW B-Students (name, sid, course)

AS SELECT S.sname, S.sid, E.cid

FROM Students S, Enrolled E

WHERE S.sid = E.sid AND E.grade = `B‘

Views, Data Independence, Security

- The *physical* schema for a relational database describes how the relations in the conceptual schema are stored, in terms of the le organizations and indexes used.
- The *conceptual schema is* the collection of schemas of the relations stored in the database.
  - While some relations in the conceptual schema can also be exposed to applications

- The view mechanism thus provides the support for *logical data independence in the relational model.*
- *That is, it can be used to define* relations in the external schema that mask changes in the conceptual schema of the database from applications.

- Views are also valuable in the context of *security*
  - We can defiene views that give a group of users access to just the information they are allowed to see

DESTROYING/ALTERING TABLES AND VIEWS

- If we decide that we no longer need a base table and want to destroy it *, we can use the DROP TABLE*
- command.