Asking Questions with Select

- *select* is used to retrieve data from one or more tables.
- `select select-list from tables`
- The special "select list *" selects all columns.
- Alternatively, the select list can be any set of SQL expressions separated by commas.
- Returns select list evaluated for all rows in table.

Simple SQL Expressions

- Column names like `title` or `checkouts` refer to the data in the current row.

- Constant expressions like `2`, `'foo bar quuux'` or `3.14159` refer to themselves.

- Arithmetic expressions like `2 * checkouts` or `position - 1` are also allowed.

Ultra-Simple Select Examples

- **`select * from books;`**

BOOKID	TITLE	SERIES	POSITION	CHECKOUTS
1	Ender's Game	Ender	1	0
2	Eon	Eon	1	20

- **`select 2.4, 2*bookid, position+bookid from books;`**

2.4	2*BOOKID	BOOKID+POSITION
2.4	2	2
2.4	4	3

Constraining Queries

- The where clause allows only a subset of the rows in a table to be retrieved.

- **`select select-list from tables where condition`**

- Condition can use simple comparisons like **`series = 'Ender'`** or **`position > 4`**.

- Parentheses, Boolean operators **`and`** and **`or`** also work.

- The **`like`** operator allows primitive regular expression matching.

- The **`distinct`** modifier eliminates duplicates.

Simple Constrained Example

- `select * from books where checkouts > 0;`

BOOKID	TITLE	SERIES	POSITION	CHECKOUTS
2	Eon	Eon	1	20

- `select title, series from books where title like '%SHIP%' and series is not null;`

TITLE	SERIES
TOM SWIFT AND HIS AERIAL WARSHIP	TOM SWIFT
TOM SWIFT AND HIS AIRSHIP	TOM SWIFT
TOM SWIFT AND HIS ROCKET SHIP	TOM SWIFT JR.
PATTON'S SPACESHIP	TIMELINE WARS
ALDAIR, MASTER OF SHIPS	ALDAIR

Sorting and the Order By Clause

- Adding a clause like **`order by column`** will sort results by that column.

- **`select title, series from books where title like '%MARS%' order by series;`**

TITLE	SERIES
MARS ARENA, THE	DEATHLANDS
SPEAR OF MARS, THE	FUTURE AT WAR
GODS OF MARS, THE	MARS
JOHN CARTER OF MARS	MARS

- More complex order-bys separating multiple keys by comma are also allowed.

- A key can be followed by **`desc`** to indicate descending order.

Cartesian Products of Tables

- Multiple tables may appear in the from-clause of a select list.
- `select series, title, author from books, authors;`
- Each row in the books table would be matched with each row in the authors table.
- If there were 100 rows in the books table and 300 rows in the authors table, this would produce $100 \times 300 = 30000$ rows.
- This is normally not useful.

Aliases

- Columns can be selected under different names.

```
select bookid as id from books;
```

ID
1
2

- Tables can be assigned extra names within queries.

```
select b.title, a.author from books b, authors a;
```

- This can be used to disambiguate queries.

```
select b.bookid, b.title, a.author from books b, authors a;
```

Equijoins

- A where clause can be used to join two or more tables on a particular column.

- ```
  select title, author from books, authors a
      where books.bookid = a.bookid;
  ```

- This query would show each book's title and author(s)—if a book has multiple authors, the book would be listed twice.

- SQL databases optimize such equijoin queries well.

- Often it helps to specify and explicit `order by` so related rows appear near each other in output.
  ```

Outer Joins

- A normal equijoin requires that matching rows exist in both tables.

- Imagine you want all book copies, and the name of the member who checked them out, but you still want to list books that are checked in. You need an outer join—a join that returns null if no match exists in one table.

- Under Oracle:

```
select b.title, m.last  from members m, books b, copies c
   where c.bookid = b.bookid and c.out_to(+) = m.memberid;
```

- Other SQL (sometimes):

```
select b.title, m.last from members m, books  b, copies c
    left outer join on c.out_to = m.memberid;
```

In, Exists and Complex Expressions

- `select first, last from members`
  `where memberid in (2021, 2228, 1013);`

- You can also use a sub-select in an *in*:

```
select first, last from members
   where memberid in (select out_to from copies
        where due_by < sysdate*)
```

- The *exists* condition can also be used:

```
select first, last from members m
    where exists (select * from copies
       where due_by < sysdate and memberid = m.memberid);
```

* In Oracle, current date

Correlation in Queries

- Last slide presents multiple ways of finding members with overdue books.

- The *in* sub-query was uncorrelated; it didn't depend on any values of the enclosing query.

- The *exists* query was correlated to the *memberid* in the enclosing query.

- Often, correlated queries are slower than uncorrelated queries.

Not and Anti-Joins

- ```
  select  title from books
       where bookid not in (
         select  bookid from copies where out_to is null);
  ```

- Reasonable optimization of anti-joins is less common than good optimization of joins.

- Other conditions like **`not exists`** also exist.

- Boolean predicates like **`not`** can be used to perform similar tasks; in general using **`not in`** and **`not exists`** may be faster.
  ```

Set operations

- The **`union operator`** combines two row sources and removes duplicates. The **`union all`** operator keeps results.

- The **`intersect`** operation returns the rows common to both sets.

- The **`minus`** operation returns the rows of the first set but not the second.

- Often, **`union all`** is faster than **`union`** because it avoids a sort.

Set Examples

- Supose we want to find all books containing "dragon" in the title but not by Piers Anthony. We could do this with an anti-join, but we won't.

- ```
  select title from books
      where title like '%DRAGON%'
  minus select title from books b, authors a
      where a.bookid = b.bookid
      and a.author = 'ANTHONY, PIERS';
  ```

- Boolean conditionals can often be used to get the same result as set operations, but set operations may perform better when the structure of the two parts of the query is significantly different.
  ```

Count and Max—Simple Group Functions

- *Group* or *set* functions retrieve statistics about groups of rows.

- The `count(*)` function returns the count of rows that match.

```
select count(*) from books
    where  title like '%DRAGON%';
```

- The `max` function find the maximum value of a particular column.

```
select max(checkouts) from books;
```

- Other group functions exist for computing minimum, average and standard deviation.

Group By and Having

- Often you wish to select statistics over *groups* of rows that share one or more columns.

- To calculate the number of books by each author:
  `select author, count(*) from authors group by author;`

- Only columns referenced in **`group by`** can be included directly in the select list.

- The **`having`** clause restricts the groups that will be returned to those meeting a condition.

Create View

- SQL provides a facility to name a select statement.

-
```
create view book_authors as
    select author, title, b.bookid from authors a, books b
    where b.bookid = a.bookid;
```

- Views can be treated as tables in the from clause of other selects.

- Views may be updatable; check your particular view to see if it works. Group functions and non-equijoins will force a view not to be updatable.

Hierarchical Queries in Oracle

- Oracle provides a facility for examining hierarchically structured data. Use this with care; it often optimizes hierarchical queries very poorly.

- The *start with* clause describes rows that are the root of a hierarchy. Like the *where* clause, *start with* can contain sub-queries.

- The *connect by* clause describes how to go down the hierarchy. It cannot contain sub-queries and must use the *prior* keyword.

- The query may include rows more than once if a row appears in the hierarchies more than once. No loops are allowed. The special variable *level* refers to the current depth in the hierarchy.

Oracle Hierarchy Example

- Construct a query to descend the tree of members based on who originally sold them their membership. Include the level in the tree.

- ```
  select level, first, middle, last
      from members start with mem_lib = first_member
      connect by prior memberid = mem_lib;
  ```
  ```

`delete` removes rows

- `delete from books` drops all data from `books` unless foreign key constraints on other tables prevent this.

- deletes may be constrained:

```
delete from books where title like '%DARK%';
delete from books where bookid in (select bookid from authors
where author = 'ANTHONY, PIERS');
```

- You can use pretty much anything that you could use in a select.

`insert` adds rows

- `insert into books values(bookid_seq.nextval*, 'ZOMBIE LOVER', 'XANTH', 23);`
  With this syntax, columns must be in the same order as the table definition. Omitted columns (`checkouts` here) get their default value or `NULL`.

- `insert into books (title, series, bookid) values ('SPEAKER FOR THE DEAD', 'ENDER', bookid_seq.nextval);`
  Here we explicitly specify what order the column values will be given in. Position has been left `NULL`.

- You can insert multiple rows at once if you use a select:
  `insert into authors (select 'ASIMOV, ISAAC' as author, b.bookid from books b where b.bookid not in (select bookid from authors));`
  makes Isaac Asimov the author for every book that doesn't have an author yet.

*this syntax is Oracle-specific

`update` changes rows

- `update authors set author = 'CARD, ORSON SCOTT' where author = 'CARD, ORSON'`
  corrects the author everywhere we forgot his middle name.

- `update copies set out_by = (select memberid from members where email = 'jsmith') where barcode = 12345;`

- The form
  `update (select ...) = ...`
  also exists, but it is very rare that you want to use this instead of simple where clauses; we couldn't come up with a counterexample in this schema.