"Relational Database Starter Day"

international php conference 2005

Lukas Kahwe Smith
smith@pooteeweet.org

Relational Database Starter Day Agenda:

- Part I
  - Getting Started
  - Retrieving Data I
- Part II
  - Data Manipulation Language (DML)
  - Data Definition Language (DDL)
- Part III
  - Retrieving Data II
- Part IV
  - Talking SQL to PHP

Getting Started: Agenda

- S[E]QL
- External and Internal Level
- Relational Algebra
- Fetching Data
- Updating Data
- Deleting Data
- Inserting Data
- Entity-Relationship Modeling
- RDBMS vs. Files and XML
- When Not To Use A RDBMS

Getting Started: S[E]QL

- Standard [English] Query Language
- Standard language to talk to Relational Database Management Systems (RDMS)
- Pronounced [SEQUEL]
- ANSI Standard
  - 1986 (SQL 87)
  - 1989 (SQL 89)
  - 1992 (SQL 92)
  - 1999 (SQL3)
  - 2003 (SQL:2003)

Getting Started: S[E]QL

- Does not cover all behavioral aspects
  - SQL actually does not cover a lot of things that people think are part of the standard!
- Is not free of ambiguity
- Often followed, rather than led, vendor implementation
- Not all vendors chose the same ways to implement the standard
  - Do not expect things to work the same on every database!

Getting Started: External Level

- Database
  - Contains Tables
    - Contains Columns and Rows
      - Specific Column-Row-Combination (Field)
- Field is the smallest retrievable unit

CONTINENTS		
CODE	NAME	Planet
E	Europe	E
NA	North America	E

CITIES		
CODE	NAME	COUNTRY
BER	Berlin	DE
FFM	Frankfurt	DE
NYC	New York	US

COUNTRIES		
CODE	NAME	CONTINENT
DE	Germany	E
US	United States	NA

Getting Started: Internal Level

- Database
  - Contains Tablespaces
    - Contains Files
      - Contains Extends
        - Contains Pages
          - Contains Rows
- Page is the smallest retrievable unit
  - Getting two rows from the same page costs the same as getting only one row!

Getting Started: Relational Algebra

- Sounds like scary math, but RDBMS are all about set theory put to practice!
- Every data row is a tuple
- Relate tuples to each other
- General set operations
  - UNION, INTERSECTION, SET DIFFERENCE, CARTESIAN PRODUCT
- SQL specific set operations
  - SELECT, PROJECT, JOIN
- The math course ends here!

Getting Started: Fetching Data

- Fetching example
  - Give me the firstname and lastname of all people where the lastname starts with a "L"
- Translates to SQL
  - SELECT firstname, lastname FROM people
    WHERE lastname = 'L%';

Getting Started: Updating Data

- Delete example
  - Delete all people where the age is lower than 18 or higher than 65
- Translates to SQL
  - DELETE FROM people
    WHERE age < 18 OR age > 65;

Getting Started: Deleting Data

- Update example
  - Update all people's lastname to 'Johnson' where the lastname is not set
- Translates to SQL
  - UPDATE people SET lastname = 'Johnson' WHERE lastname IS NULL;

Getting Started: Inserting Data

- Inserting example
  - Add a new person with the values 'Don', 'Johnson' and 47
- Translates to SQL
  - INSERT INTO people
    VALUES ('Don', 'Johnson', 47);

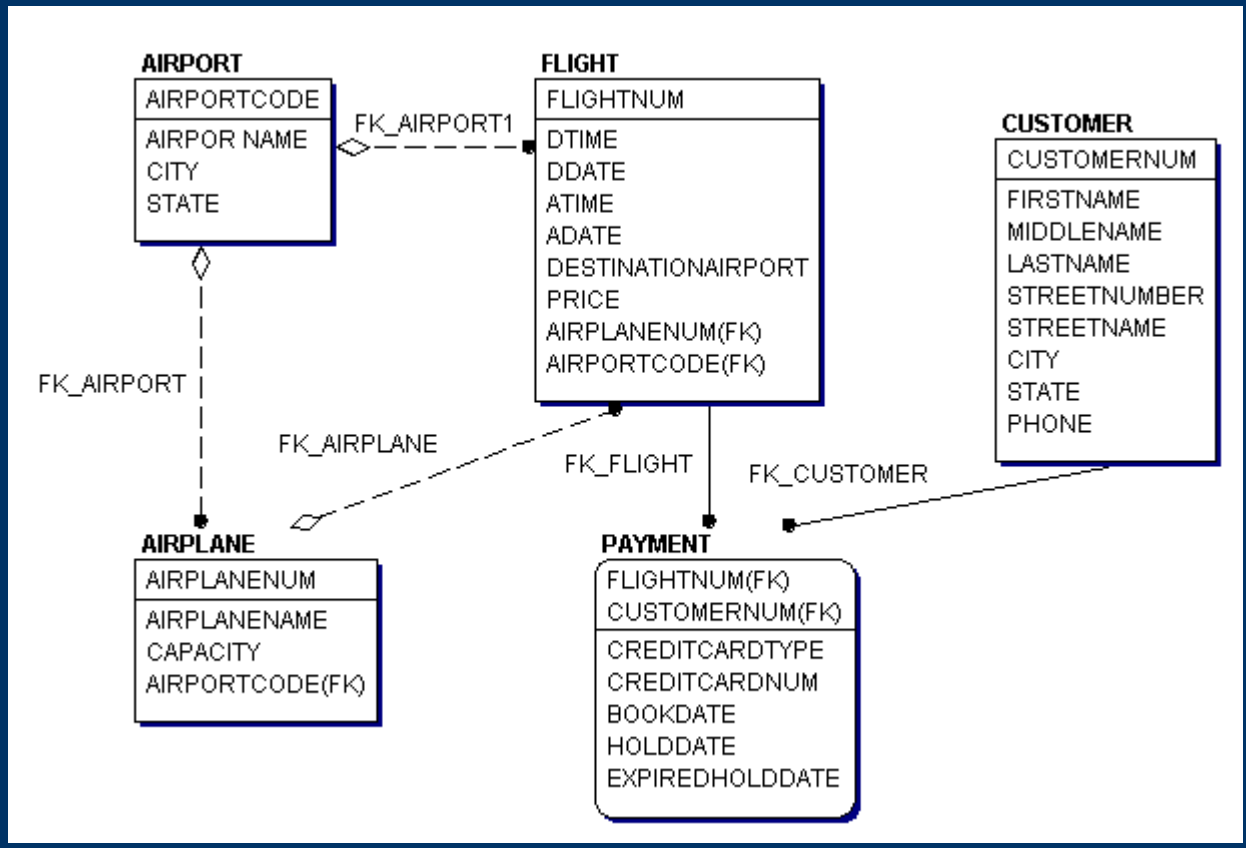
Getting Started: Entity-Relationship Modeling

- Method of establishing and visualizing the structure of a database
- Entity-Relationship (ER) Diagrams for schema modeling using
  - Entities
  - Attributes
  - Relationships
- Enhanced-Entity-Relationship (EER) diagrams incorporate additionally
  - Class/subclass relationship
  - Type inheritance

Getting Started: Entity-Relationship Modeling

- Identifying Entities
  - Correspond roughly to tables
- Identifying Attributes
  - Correspond roughly to columns in tables
- Identifying Relationships
  - One-to-One Relationships
    - Usually do not require a separate table
  - Many-to-One Relationships
    - Simple relationship between two columns
  - Many-to-Many Relationships
    - Require separate relationship table

Getting Started: Entity-Relationship Modeling



Taken from: http://staffwww.fullcoll.edu/brippe/461/docs/ERDiagram.gif

Getting Started: RDBMS vs. Files

- Defined industry standard interface
- Optimized access to data
- Finely controlled access to data
- Automatic constraint checking
- Ability to relate data instead of duplicating it

Getting Started: RDBMS vs. XML

- No need to store redundant data
- Fast direct access to all data
- Fixed structure, not dynamically extensible
- Hierarchical structures hard to model and efficient retrieval only possible through proprietary extensions

Getting Started: When Not To Use A RDBMS

- Requirements are simple and fixed
- Absolute maximum performance is of the highest importance
- No ability to define a strict structure

Retrieving Data I: Agenda

- SELECT
- Select List
- WHERE
- ORDER BY
- DISTINCT
- SQL Functions

Retrieving Data I: SELECT

- Following statement
  - Give me all columns about people
- Translates to SQL
  - SELECT * FROM people
- "SELECT" – get data
- "*" - call columns
- "FROM" – start table listing
- "people" – the table to select from

Retrieve Data I: SELECT

- Returned data from a SELECT is called "result set" or "rowset"
- Qualify names with a "." within schema
  - SELECT people.lastname FROM people
    WHERE mydb.people.lastname = 'Smith'
- Control flow and conditional statements
  - SELECT CASE age WHEN 1 THEN 'one'
    WHEN 2 THEN 'two'
    ELSE 'more' END;
  - SELECT COALESCE(NULL, 1);
  - SELECT NULLIF(NULL, 1);

Retrieving Data I: Select List

- "*" is evil
  - What if someone changes the order of the columns or adds a new column in the table?
  - Convenient on the CLI or workshop slides ;)
- List names of columns in required order
  - SELECT firstname, lastname FROM people
- "firstname, lastname" – select list
- Alias column (and table) names
  - SELECT firstname, firstname AS name, firstname AS name2 FROM people
  - SELECT firstname FROM people AS persons

Retrieving Data I: WHERE

- Filter the result set
  - SELECT * FROM people
    WHERE firstname = 'Lukas'
- "WHERE" – begin filter rules
- "firstname" – name of the column
- "=" - operator
- "'Lukas'" – literal
  - Literal strings must be quoted

Retrieving Data I: WHERE

- May compare columns to columns
  - SELECT * FROM people
    WHERE firstname = lastname
- And literals to literals
  - SELECT * FROM people WHERE 1 = 0

Retrieving Data I: WHERE

- Wide range of boolean operators: =, >, <, =>, =<, !=, <>, NOT, IN, BETWEEN
  - WHERE firstname = 'Lukas' AND lastname = 'Smith'
  - WHERE age => 29 OR firstname <> 'Lukas'
  - WHERE age < 28 AND age > 30
  - WHERE lastname != 'SMITH'
  - WHERE NOT lastname = 'Smith'
  - WHERE firstname IN ('Lukas', 'Kahwe')
  - WHERE age BETWEEN 20 AND 30

Retrieving Data I: WHERE

- Boolean pattern matching operators: LIKE, SIMILAR TO, REGEXP, MATCH
  - WHERE lastname LIKE '_m%'
    - "_" - some character
    - "%" - any number of characters
  - WHERE firstname SIMILAR TO 'Derick|rick'
  - WHERE firstname REGEXP '[De]?rick'
  - WHERE MATCH (comment) AGAINST ('+SQL -rational' IN BOOLEAN MODE);

Retrieving Data I: WHERE

- Wide range of mathematical operators: %, /, *, +, -, &, |
  - SELECT * FROM items
    WHERE (price * total) > 60
  - SELECT (total + 5)
    AS total_with_shipping FROM items
  - SELECT (is_active & is_new) FROM items

Retrieving Data I: WHERE

- Use parenthesis to group clauses
- Operator precedence
  - Parenthesis bind strongest
  - Mathematical operators
    - Multiplications
    - Other
  - Boolean Operators
  - AND binds stronger than OR
  - SELECT * FROM people
    WHERE firstname = 'Lukas' AND
    (age > 18 OR lastname = 'Smith')

Retrieving Data I: ORDER BY

- Order the result set by columns
  - SELECT * FROM people
    ORDER BY lastname, firstname
- "ORDER BY" – start order list
- "lastname, firstname" – order list
- Combine ORDER BY with WHERE
  - SELECT * FROM people
    WHERE firstname = 'Lukas'
    ORDER BY lastname

Retrieving Data I: DISTINCT

- Filter duplicate rows from the result set
  - SELECT DISTINCT firstname FROM people
    WHERE lastname = 'SMITH'
- "DISTINCT" – filter out duplicates

Retrieving Data I: SQL Functions

- Apply a wide range of functions
  - SELECT CONCAT(age, '!') FROM people WHERE SUBSTRING(firstname FROM 1 FOR 1) = 'L'
- "CONCAT()" – concatenation function
- "SUBSTRING()" – substring function
- Various functions are implemented in most RDBMS
- Language is not as rich as most common programming languages

End of Part I
Questions?

Data Manipulation Language: Agenda

- INSERT
- UPDATE
- DELETE
- REPLACE
- LOAD, COPY
- TRUNCATE

Data Manipulation Language: INSERT

- Simple INSERT statement
  - INSERT INTO people
    (firstname, lastname, age) VALUES
    ('Lukas', 'Smith', 27)
- "INSERT INTO" – insert data
- "people" – table to insert into
- "(firstname, lastname, age)" - column list, optional columns may be omitted
- "VALUES" - start listing of data
- ('Lukas', 'Smith', '28')" - data to insert

Data Manipulation Language: My Birthday

It was my birthday!!!!
... yesterday

Data Manipulation Language: UPDATE

- Simple UPDATE statement
  - UPDATE people SET firstname = 'Lukas', age = 28 WHERE lastname = 'Smith'
- "UPDATE" - update data
- "people" - table in which to update
- "SET" - begin update listing
- "firstname = 'Lukas', age = 28" - new column values to set
- "WHERE" – begin filter rules
- "lastname = 'Smith'"- filter rules

Data Manipulation Language: DELETE

- Simple DELETE statement
  - DELETE FROM people
    WHERE lastname = 'Smith'
- "DELETE" - delete data
- "FROM" – start table listing
- "people" - table in which to delete
- "WHERE" – begin filter rules
- "lastname = 'Smith'"- filter rules

Data Manipulation Language: REPLACE

- Not standard SQL syntax
- Simple REPLACE statement
  - REPLACE INTO people
    (user_id, firstname, lastname, age)
    VALUES (23, 'Lukas', 'Smith', 28)
- "REPLACE INTO" – insert or update data
- Works like INSERT but deletes row with same PRIMARY KEY or UNIQUE INDEX
- Only supported by MySQL and SQLite

Data Manipulation Language: LOAD, COPY ..

- Not standard SQL syntax
- Bulk inserting of data from/to a file
- Radically faster than multiple inserts
- Simple COPY statement
  - COPY people FROM 'c:/tmp/people' USING DELEMITERS '|' WITH NULL AS '';
- Every RDBMS seems to have come up with a syntax of their own

Data Manipulation Language: TRUNCATE

- Not standard SQL syntax
- Deletes all rows from a table in the most efficient manner
- Simple TRUNCATE statement
  - TRUNCATE TABLE people
- "TRUNCATE TABLE" - delete all data
- "people" - table in which to delete
- Most RDBMS support this syntax or just use a DELETE-statement
  - DELETE FROM people

Some playing time ..

Data Definition Language: Agenda

- Normalization
- Tables
- NULL
- Views
- Indexes
- Constraints
- Stored Procedure
- Trigger

Data Definition Language: Normalization

- Process of removing redundant data
- Normalization helps avoid INSERT UPDATE and DELETE anomalies
- Anomalies cause
  - INSERT to require unnecessary data
  - DELETE remove too much data
  - UPDATE to deal with redundant data
- Multiple normal forms (NF) exist
- Most databases are third normal form
- There are even stricter normal forms

Data Definition Language: Normalization

- PRIMARY KEY uniquely identifies a row
- CANDIDATE KEY is any set of columns that uniquely identify a row
- Example non normalized table
  - Name is the PRIMARY KEY
  - Salary could be considered a CANDIDATE KEY for this set of data

Diplomats					
Name	Title	Language	Salary	Work_group	Head_honcho
Axworthy	Consul	French, German	30,000.00	WHO, IMF	Greene, Craig
Broadbent	Diplomat	Russian, Greek	25,000.00	IMF, FTA	Craig, Crandall
Craig	Ambassador	Greek, Russian	65,000.00	IMF	Craig
Crandall	Ambassador	French	55,000.00	FTA	Crandall
Greene	Ambassador	Spanish, Italian	70,000.00	WHO	Greene

Data Definition Language: Normalization

- First Normal Form or 1NF
  - All columns only contain only scalar values
    - as opposed to lists of values
  - Name, Language and Work_group are now the PRIMARY KEY

Diplomats

Name	Title	**Language**	Salary	**Work_group**	Head_honcho
Axworthy	Consul	French	30,000.00	WHO	Greene
Axworthy	Consul	German	30,000.00	IMF	Craig
Broadbent	Diplomat	Russian	25,000.00	IMF	Craig
Broadbent	Diplomat	Greek	25,000.00	FTA	Crandall
Craig	Ambassador	Greek	65,000.00	IMF	Craig
Craig	Ambassador	Russian	65,000.00	IMF	Craig
Crandall	Ambassador	French	55,000.00	FTA	Crandall
Greene	Ambassador	Spanish	70,000.00	WHO	Greene
Greene	Ambassador	Italian	70,000.00	WHO	Greene

Data Definition Language: Normalization

- One group of values is functionally dependent on another if for
  - every possible value set in the first group
  - there is one and only one possible value set for the items in the second group
- Must hold true for all possible values

PRODUCTS		
NAME	PRICE	COLOR
Table	78.00	White
Table	78.00	Black
Chair	14.00	White
Table	78.00	Green

Data Definition Language: Normalization

- Second Normal Form or 2NF
  - All nonkey columns in a table must be functionally dependent on the primary key

Diplomats

Name	Title	Salary	**Work_group**	Head_honcho
Axworthy	Consul	30,000.00	WHO	Greene
Axworthy	Consul	30,000.00	IMF	Craig
Broadbent	Diplomat	25,000.00	IMF	Craig
Broadbent	Diplomat	25,000.00	FTA	Crandall
Craig	Ambassador	65,000.00	IMF	Craig
Crandall	Ambassador	55,000.00	FTA	Crandall
Greene	Ambassador	70,000.00	WHO	Greene

Languages

Name	**Language**
Axworthy	French
Axworthy	German
Broadbent	Russian
Broadbent	Greek
Craig	Greek
Craig	Russian
Crandall	French
Greene	Spanish
Greene	Italian

Data Definition Language: Normalization

- Third Normal Form or 3NF
  - All nonkey columns in a table must be functionally dependent on a candidate key

Diplomats

Name	Title	Salary
Axworthy	Consul	30,000.00
Broadbent	Diplomat	25,000.00
Craig	Ambassador	65,000.00
Crandall	Ambassador	55,000.00
Greene	Ambassador	70,000.00

Affiliations

Name	**Work_group**
Axworthy	WHO
Axworthy	IMF
Broadbent	IMF
Broadbent	FTA
Craig	IMF
Crandall	FTA
Greene	WHO

Languages

Name	**Language**
Axworthy	French
Axworthy	German
Broadbent	Russian
Broadbent	Greek
Craig	Greek
Craig	Russian
Crandall	French
Greene	Spanish
Greene	Italian

Work_groups

Work_group	Head_honcho
WHO	Greene
IMF	Craig
FTA	Crandall

Data Definition Language: Normalization

- Intentionally violating normalization rules is sometimes feasible
  - Improve performance
  - Simplify queries
    - For example storing the age computed from the birth date in a separate column
- Application needs to handle anomalies
- RDBMS provide different features to automatically handle denormalization
  - Indexes
  - Materialized Views

Data Definition Language: Tables

- Creating a table
  - CREATE TABLE people (
    people_id UNSIGNED INT NOT NULL,
    firstname VARCHAR(200),
    lastname VARCHAR(200),
    age SMALLINT NOT NULL DEFAULT 0,
    PRIMARY KEY (people_id)
)
- Contains list of all columns, their data types, defaults and optional modifiers

Data Definition Language: Tables

- Data types include multiple
  - String types (CHAR, VARCHAR)
  - Numeric types (INT, DECIMAL, FLOAT)
  - Temporal types (TIMESTAMP, DATE, TIME)
  - Specialized types (like spatial types)
- Choosing the proper data type is important to get optimal performance
- Access to fixed length fields is faster
- Access to integer fields is faster
- String data types require choosing a charset for storage and sorting

Data Definition Language: Tables

- Dropping a table
  - DROP TABLE people
- Most RDBMS allow (some) changes to an existing table structure
  - ALTER TABLE people DROP age,
    ADD bday TIMESTAMP NOT NULL
  - ALTER TABLE people RENAME TO persons

Data Definition Language: Notes on NULL

- NULL means undefined or unknown
- Every NULL value is considered to be distinct from another NULL value
  - some operators break this rule
  - NULL = NULL // evaluates to false
  - NULL IS NULL // evaluates to true
  - NULL IS NOT NULL // evaluates to false
  - 'Lukas' IS NULL // evaluates to false
- Allowing NULL requires extra space
- Some RDBMS make NULLable columns variable length

Data Definition Language: Views

- Splitting off a new "Conceptual Level" from the "External Level"
- Virtual tables generated from a SELECT
  - Convenience
  - Finely grained access for users
  - CREATE VIEW people_age AS
    SELECT firstname, lastname,
    TIMESTAMPDIFF(YEAR, bday,
    CURRENT_TIMESTAMP) AS age
    FROM people
  - DROP VIEW people_age

Data Definition Language: Views

- VIEW supports SELECT
- VIEW supports DML if
  - There is a 1-1 mapping between the VIEW and the underlying TABLE
    - No mathematical expressions and aggregates in the select list
    - No DISTINCT
    - No GROUP BY and HAVING
    - No references to multiple tables (ANSI)
    - No functional calls in the select list (ANSI)
    - No references to non-updateable view in the select list (ANSI)

Data Definition Language: Indexes

- Indexes not part of the SQL standard
- Speed up searches
  - May be thought of as DBMS managed denormalized partial tables
- RDBMS may choose to use the index
  - instead of searching the actual table
  - instead of reading from the actual table
    - the latter is called a covering index
- Multiple indexes are allowed per table
- May be specified on multiple columns within a given table

Data Definition Language: Constraints

- Constraints ensure data integrity
- UNIQUE and PRIMARY KEY may be specified on multiple columns within a given table
  - allow no duplicates
  - PRIMARY KEY columns do not all NULL
  - usually implemented using an index
    - Avoid redundant indexes
  - Multiple UNIQUE KEY allowed per table
  - Only one PRIMARY KEY is allowed per table

Data Definition Language: Constraints

- FOREIGN KEY specify that the given column references another column
  - Use of ON [UPDATE|DELETE] CASCADE enables automatic update/cleanup
  - FOREIGN KEY REFERENCES people (people_id) ON DELETE CASCADE
- CHECK constraint allows specification of additional column value criteria
  - CHECK (age > 18)

Data Definition Language: Stored Procedure

- Definition of complex business logic within the database
  - Associate logic closely to data
  - No need for additional layers
  - Better performance, less network traffic
  - Integrated with RDBMS access controls
  - CREATE PROCEDURE hello_world()
    SELECT 'Hello World'
  - CALL hello_world()
  - DROP PROCEDURE hello_world
- Usually support loops, if/else, case etc.

Data Definition Language: Trigger

- Specialized stored procedure that automatically executes on events
- Used to enforce data integrity or to run logic on INSERT, UPDATE or DELETE
  - CREATE TRIGGER check_age ON people FOR INSERT AS BEGIN IF age < 18 BEGIN ROLLBACK TRANSACTION PRINT "age must be above 18." END END
  - DROP TRIGGER check_age

Some playing time ..

End of Part II

Questions?

Retrieving Data II: Agenda

- LIMIT
- Aggregate Functions
- GROUP BY
- UNION
- JOIN
- Subqueries
- JOIN/Subqueries and DML
- Optimizer
- EXPLAIN
- Transactions

Retrieving Data II: LIMIT

- Web applications often require paged views of result sets
- Standard approach
  - SELECT * FROM (SELECT ROW_NUMBER() OVER (ORDER BY key ASC) AS rownum, * FROM people) AS foo WHERE rownum > skip AND rownum <= (n+skip)
- MySQL, PostGreSQL, SQLite
  - SELECT * FROM people
    ORDER BY key ASC LIMIT n OFFSET skip
- Alternatively CURSORs may be used

Retrieving Data II: Aggregate Functions

- Act on value sets returned by a query
- All aggregate functions ignore NULLs
  - COUNT()
    - Count the number of columns
  - MIN()/MAX()
    - Return the minimal/maximal value of a column
  - AVG()
    - Return the mean value of a column
  - SUM()
    - Return the sum of all columns
  - SELECT MIN(price) FROM items

Retrieving Data II: GROUP BY

- Categorizes rows in a result set by the contents of one or more columns
- Combine with other select clause items
  - WHERE is evaluated before the GROUP BY
  - Only grouped columns or aggregates allowed in the select list
  - SELECT cat_id, SUM(price) FROM items
    WHERE price > 30
    GROUP BY cat_id
    ORDER BY cdate

Retrieving Data II: GROUP BY

- Some overlap with DISTINCT operator
  - SELECT DISTINCT id FROM items
  - SELECT id FROM items GROUP BY id
- With HAVING the result set can be filtered after the GROUP BY
  - Only elements from the select list may be included in the HAVING clause
  - SELECT cat_id, SUM(price) FROM items
    GROUP BY cat_id
    HAVING SUM(price) > 50

Retrieving Data II: UNION, INTERSECT, MINUS

- UNION combines the result sets of multiple separate SELECT statements
  - Requires compatible select lists
    - Same number of columns
    - Same (or similar) column data types
    - SELECT name, cdate FROM foo UNION SELECT lastname, cdate FROM bar
    - UNION discards all duplicate rows
    - Using UNION ALL all duplicates are retained
- INTERSECT returns intersection
- MINUS returns difference

Retrieving Data II: JOIN

- JOIN allows working on data from multiple tables in a single statement
- Produces cartesian cross of table rows
- Old style with JOIN condition
  - SELECT * FROM t1, t2
    WHERE t1.col1 = t2.col1 AND age > 18
  - NULL values are discarded in the JOIN
- SELF JOIN joins a table with itself
  - SELECT * FROM orders AS o1, orders AS o2
- Any number of tables can be joined
  - SELECT * FROM t1, t2, t3

Retrieving Data II: JOIN

- ANSI style without a JOIN condition
  - SELECT * FROM t1 CROSS JOIN t2
- ANSI style with JOIN condition
  - SELECT * FROM t1 JOIN t2
    ON t1.col1 = t2.col1 WHERE ..
  - SELECT * FROM t1 INNER JOIN t2
    ON t1.col1 = t2.col1 WHERE ..
  - SELECT * FROM t1 JOIN t2
    USING (col1) WHERE ..
  - SELECT * FROM t1 NATURAL JOIN t2
    WHERE ..

Retrieving Data II: JOIN

- ANSI Style more clearly separates the JOIN condition from a WHERE clause
  - SELECT * FROM people JOIN accounts
    ON people.user_id = account.user_id
    WHERE funds > 600.00
- ANSI Style allows LEFT and RIGHT OUTER JOIN to include all records from the left/right table of the operator
  - SELECT * FROM people
    LEFT OUTER JOIN accounts
      ON people.user_id = accounts.user_id

Retrieving Data II: Subqueries

- SELECT statement contained within another statement
  - SELECT * FROM people WHERE user_id IN (SELECT user_id FROM accounts);
- May return a single field, a column or a number of rows
  - WHERE (user_id, jdate) = (SELECT user_id, cdate FROM orders);
- May be used inside the FROM clause
  - SELECT * FROM people, (SELECT cdate FROM orders) AS orders

Retrieving Data II: Subqueries

- EXISTS returns true if the subquery returns any values
  - SELECT * FROM people WHERE
    EXISTS (SELECT * FROM items) AND
    NOT EXISTS (SELECT * FROM accounts)
- ANY, ALL allow comparison with other operators than equality with "=" or "IN"
  - SELECT * FROM people WHERE 60 <= ANY
    (SELECT SUM(price) FROM order
    WHERE order.user_id = people.user_id
    GROUP BY order_id)

Retrieving Data II: JOIN/Subqueries and DML

- JOIN and Subqueries may be used within DML statements
  - UPDATE items, accounts
    SET items.price = accounts.funds
    WHERE accounts.user_id = 4
  - DELETE FROM accounts WHERE user_id IN (SELECT user_id FROM people)

Retrieving Data II: Optimizer

- Rule-based optimizers use non volatile data and fixed assumptions
- Cost-based optimizers additionally use table statistics and other volatile data
  - Everybody claims to be cost-based
  - Biggest advantage for cost-based optimizers is for joins
- Statistics may change over time
  - Use ANALYZE, OPTIMIZE, VACUUM or some other RDBMS specific command to keep tables in mint condition

Retrieving Data II: EXPLAIN

- Show execution plan for a given query
  - How will the table(s) be scanned?
  - What indexes will be used?
  - What join algorithms will be used?
  - What is the estimated „execution cost"?
- Tool of choice for query optimizations
- Not part of the SQL standard
- All DBMS have some equivalent
  - SET EXPLAIN, SELECT .. PLAN, etc.

Retrieving Data II: Transactions

- Enforce handling of multiple queries as one atomic unit that can be undone
  - BEGIN
    - SELECT price FROM items WHERE item_id = 23
    - SELECT funds FROM credit WHERE user_id = 2
    - UPDATE account SET funds = funds – 20.45 WHERE user_id = 2
    - INSERT INTO orders VALUES (2, 23)
  - COMMIT
  - Use ROLLBACK at any point in the transaction to undo the entire transaction
- DDL statements may implicitly COMMIT

Some playing time ..

End of Part III

Questions?

Talking SQL to PHP: Agenda

- PDO Extension and PEAR::MDB2
- Installation
- Reasons for Database Abstraction
- Connecting
- Data Manipulation
- Data Fetches
- Prepared Statements
- Error Handling
- Transactions
- Advanced PEAR::MDB2 Usage

Talking SQL to PHP: PDO Extension

- C Extension
- OO Interface
- Driver model
- Client API Unification
- Bundled in PHP 5.1, PHP 5.x compatible
- High Performance
- Optionally supports exceptions
- Prepared statements, Cursors, Iterators
- Limited set of portability modes

Talking SQL to PHP: PEAR::MDB2

- Written in PHP
- OO Interface
- Driver model
- Compatible with PHP4 and PHP5
- Wide range of optional portability modes
- On demand loading of
  - Data type abstraction
  - Schema management abstraction
  - Schema reverse engineering
  - Function call abstraction

Talking SQL to PHP: PEAR::MDB2

- Mixes the PEAR::DB and PDO API
- PEAR_Error and
- Internal debug capabilities
- Prepared Statements, Iterators
- XML based schema abstraction using PEAR::MDB2_Schema generates
  - CREATE, DROP, ALTER, INSERT statements

Talking SQL to PHP: Installation

- Old "Native" Extensions
  - Compile --with-[RDBMS][=DIR]
- PDO Extensions
  - Compile --with-zlib --enable-pdo=shared
  - pear install pdo-[state]
  - pear install pdo_[RDBMS]-[state]
- Or grab from http://pecl4win.php.net
- Enable extensions in php.ini
- MDB2 php classes
  - pear install MDB2-[state]
  - pear install MDB2_[RDBMS]-[state]

Talking SQL to PHP: Reason for Database Abstraction

- Support for multiple RDBMS
  - Save time by only hand tweaking real world bottlenecks
- Forward compatibility
  - new RDBMS versions
  - new PHP (extension) versions (mysqli/PDO)
- Prevent vendor Lock in
- Reduced train costs
- Pushing own preference to the client
- Get a higher level API

Talking SQL to PHP: Connecting

- Connecting to PDO

```
// odbc:odbc_dsn
// mysql:host=name;dbname=dbname
// sqlite:/path/to/db/file
$dsn = $rdbms_specific;
try {
    $pdo = new PDO($dsn,
      $user, $password, $options);
} catch (PDOException $e) {
    echo "Failed to connect:"
       . $e->getMessage();
}
```

Talking SQL to PHP: Connecting

- Connecting to MDB2

```
$dsn = array(
    'phptype'  => $rdbms,
    'username' => 'foo',
    'password' => 'bar',
    'database' => 'mydb'
);
$mdb2 =& MDB2::factory($dsn, $options)
if (PEAR::isError($mdb2)) {
   echo "Failed to connect:"
       . $mdb2->getMessage();
}
```

Talking SQL to PHP: Data Manipulation

- PDO

```
// some RDBMS do not return affected
// rows unless you do "WHERE 1"
$sql = "DELETE FROM foo WHERE 1";
$affected = $pdo->exec($sql);
```

- MDB2 (will migrate to the PDO way)

```
$sql = "UPDATE foo SET age = 28 WHERE
  name = ".$mdb2->quote('Smith');
$affected = $mdb2->query($sql);
```

Talking SQL to PHP: Data Manipulation

- PDO (also works in MDB2)

```
$sql = "INSERT INTO $table ('foo')";
$pdo->exec($sql);
$id = $pdo->lastInsertId($table);
```

- MDB2 also supports

```
$id = $mdb2->extended->
  getBeforeId($table);
$sql = "INSERT INTO $table (".$mdb2->
  quote($id, 'integer').', 'foo')";
$mdb2->query($sql);
$id = $mdb2->extended->getAfterId(
  $id, $table);
```

Talking SQL to PHP: Data Fetching

- PDO provides fetch methods on the statement object

```
$stmt = $pdo->query($sql);
while ($row = $stmt->fetch())
    print_r($row);
```

- MDB2 provides fetch methods via a separate result set object

```
$result = $mdb2->query($sql);
while ($row = $result->fetchRow())
    print_r($row);
```

Talking SQL to PHP: Data Fetching

- PDO and MDB2 fetch modes
  - Array with numeric and string keys
  - Array with numeric keys
  - Array with string keys
  - Object with properties
  - Fetch a numbered column only
  - Fetch into a specific object or existing object instance
- PDO only
  - Just returns true until end of result set
  - Passes row to a callback function

Talking SQL to PHP: Data Fetching

- PDO defaults to unbuffered result sets
- MDB2 defaults to buffered result sets
- MDB2 supports unbuffered result sets

```
$mdb2->setOption(
  'result_buffering', false);
```

- Buffered result sets fetch the entire result set in the client before returning
- Buffering takes time and memory
- Buffering frees up resources on the server quickly

Talking SQL to PHP: Data Fetching

- PDO and MDB2 support reading all rows into a multi dimensional array
  - PDO's way to support result buffering
  - Using "rekey" to get first column as the key for the first array dimension
  - Using "group" to have all rows with the same value in the first column grouped inside another array

```
$result = $mdb2->query($sql);
$array = $result->fetchAll(
  $mode, $rekey, $force_array, $group);
```

Talking SQL to PHP: Data Fetching

- Bind result set values to variables

```
$sql = "SELECT foo, bar FROM foobar"
$stmt = $pdo->prepare($sql);
$stmt->execute();
$stmt->bindColumn('foo', $foo);
$stmt->bindColumn('bar', $bar);
while ($stmt->fetch(PDO_FETCH_BOUND))
  echo "Foo: $foo, Bar: $bar\n";
```

Talking SQL to PHP: Data Fetching

- PDO and MDB2 support a wide selection of portability aids for result sets
  - NULLs vs. empty strings
  - Case and database/table qualified names (MDB2 only) in associative fetches
  - Padded strings (MDB2 only)

```
$pdo->setAttribute(
  PDO_ATTR_CASE, PDO_CASE_UPPER);
$sql = "SELECT bar FROM foo";
$stmt = $pdo->prepare($sql);
$stmt->execute();
$stmt->bindColumn('BAR', $bar);
```

Talking SQL to PHP: Data Fetching

- PDO and MDB2 support iterators for result sets

```
$sql = "SELECT bar FROM foo";
$stmt = $pdo->query(
  $sql, PDO_FETCH_COLUMN, 0);
foreach ($stmt as $bar) {
   echo "Bar: $bar\n";
}

$mdb2->setOption('result_wrap_class',
  MDB2_BufferedIterator);
$result = $mdb2->query($sql);
```

Talking SQL to PHP: Data Fetching

- PDO supports CURSOR
  - Random access to rows in the result set
  - Useful to emulate LIMIT paged queries
  - Enable positioned updates

```
$sql = "SELECT * FROM foo";
$attr = array(
  PDO_ATTR_CURSOR =>PDO_CURSOR_SCROLL);
$stmt = $pdo->prepare($sql, $attr);
```

- MDB2 supports LIMIT

```
// set limit to 10 and offset to 20
$mdb2->setLimit(10, 20);
$result = $mdb2->query($sql);
```

Talking SQL to PHP: Prepared Statements

- Bind values to SQL placeholders

```
// create sql string
$sql = 'INSERT INTO foo (:url,:file)';
$types = array('text', 'blob')
$stmt = $mdb2->prepare($sql, $types);
// bind values
$stmt->bindParam(‘url', $text);
$stmt->bindParam(‘file', $fp);
$result = $stmt->execute();
// bind next set of data
$text = 'foo'; $fp = 'bar.pdf';
$result = $stmt->execute();
```

Talking SQL to PHP: Prepared Statements

- PDO supports IN/OUT parameters

```
$sql = "call @sp_inout(?, ?)";
$stmt = $pdo->prepare($sql);
$val = "My Input Data";
// notice that PDO is 1-indexed,
// while MDB2 is 0-indexed
$stmt->bindParam(1, $val,
  PDO_PARAM_STR|PDO_PARAM_INPUT_OUTPUT,
  4000);
$stmt->bindParam(2, $val)
if ($stmt->execute()) {
    echo "Got $val\n"; }
```

Talking SQL to PHP: Error handling

- PDO supports multiple error handler and provides SQLSTATE error codes

```
// PDO_ERRMODE_SILENT,
// PDO_ERRMODE_WARNING,
// PDO_ERRMODE_EXCEPTION
$pdo->setAttribute(PDO_ATTR_ERRMODE,
  PDO_ERRMODE_SILENT);
if (!$pdo->query($sql)) {
 echo $pdo->errorCode();
 // $info contains SQLSTATE/native
 // error code and native errormessage
 $info = $pdo->errorInfo(); .. }
```

Talking SQL to PHP: Error handling

- PEAR based error handling in MDB2
- Overwrite global handler locally

```
PEAR::setErrorHandling(..);
// expect no such table error
$mdb2->expectError(
  MDB2_ERROR_NOSUCHTABLE);
// send off query
$result = $mdb2->query($sql);
$mdb2->popExpect();
// table did not exist?
if (PEAR::isError($result,
  MDB2_ERROR_NOSUCHTABLE) { .. }
```

Talking SQL to PHP: Transactions

- Optionally use auto commit mode or transactions

```
$pdo->setAttribute(
  PDO_ATTR_AUTOCOMMIT, false);
$pdo->beginTransaction();
try {
    $pdo->query($sql1);
    $pdo->exec($sql2);
    $pdo->commit();
} catch (Exception $e) {
    $pdo->rollBack();
}
```

Talking SQL to PHP: Advanced PEAR::MDB2 Usage

- Data type abstraction

```
$mdb2->quote($text, 'text');
$mdb2->quote($date, 'timestamp');
$mdb2->quote($bool, 'boolean');
$mdb2->quote($fp, 'blob');

$row = $mdb2->query($sql,
  array('boolean', 'timestamp'));
if ($row['bool'] === true) { .. }
// create custom abstract data types
$mdb2->quote($obj, 'serialize');
$obj = $mdb2->query($sql,'serialize');
```

Talking SQL to PHP: Advanced PEAR::MDB2 Usage

- Manager module for DDL abstraction
- Handles database, table, index, constraint and sequence management

```
$mdb2->loadModule('Manager');
$mdb2->manager->createTable(..);
$mdb2->manager->listTables(..);
$mdb2->manager->alterTable();
$mdb2->manager->dropTable();
..
```

Talking SQL to PHP: Advanced PEAR::MDB2 Usage

- Reverse module for schema reading

```
$mdb2->loadModule('Reverse');
$mdb2->reverse->
  gettableFieldDefinition(..);
$mdb2->reverse->
  getTableIndexDefinition(..);
$mdb2->reverse->
  getTableConstraintDefinition(..);
$mdb2->reverse->
  getSequenceDefinition(..);
$mdb2->reverse->tableInfo(..);
```

Talking SQL to PHP: Advanced PEAR::MDB2 Usage

- Function call abstraction

```
// load function module
$mdb2->loadModule('Function');
// Oracle would return " FROM dual"
$funcTable = $mdb2->function->
  functionTable();
$nowClause = $mdb2->function->
  now('timestamp');
// SELECT CURRENT_TIMESTAMP FROM dual
$sql = "SELECT $nowClause$funcTable";
$result = $mdb2->queryOne($sql);
```

Talking SQL to PHP: Advanced PEAR::MDB2 Usage

- Replace emulation
- Subselect emulation

```
$sql = "SELECT name FROM foo";
// returns or executes $query
$subselect = $mdb2->subSelect(
  $sql, 'text');
// sql string
$sql = "SELECT is_active FROM bar
  WHERE name IN ($subselect)";
// send off query
$res = $mdb2->query($sql, 'boolean');
```

Talking SQL to PHP: Advanced PEAR::MDB2 Usage

- PEAR::MDB2_Schema provides XML based schema definition
- Generate CREATE , ALTER, DROP, INSERT statements
- Ability to compare schemas
  - Databases
  - Tables
    - Columns
    - Indexes
    - Constraints
  - Sequences/Autoincrement

Some playing time ..

References:

- This workshop
  - http://pooteeweet.org/files/phpconf05/relational_database_starter_day.pdf
- MySQL Sample Database
  - http://www.openwin.org/mike/index.php/archives/2005/09/the-sample-db-moves-on/

References:

- Books
  - "SQL Performance Tuning"
    - by Peter Gulutzan and Trudy Pelzer
  - "Fundamentals of Database Systems"
    - by Ramez Elmasri and Shamkant Navathe
  - "Mastering SQL"
    - by Rafe Colburn
  - "SQL-99 Complete, Really"
    - by Peter Gulutzan and Trudy Pelzer

References:

- Other internet resources
  - http://oss.backendmedia.com
    - PDO and MDB2 wiki by Lukas Smith
  - http://netevil.org/talks/?t=pdo
    - PDO talk by Wez Furlong
  - http://troels.arvin.dk/db/rdbms/
    - SQL Syntax differences by Troels Arvin

Thank you for listening ..
Comments? Questions?

smith@pooteeweet.org